Modelling and Analysing Access Control Policies in XACML 3.0

Ramli, Carroline Dewi Puspa Kencana

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Modelling and Analysing Access Control Policies in XACML 3.0

Carroline Dewi Puspa Kencana Ramli

Kongens Lyngby 2015
PhD-2015-364

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, Building 324,
DK-2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk PhD-2015-364

Summary

XACML (eXtensible Access Control Markup Language) is a prominent access control language that is widely adopted both in industry and academia. XACML is an international standard in the field of information security. The problem with XACML is that its specification is described in natural language (c.f. [GM03, Mos05, Ris13]) and manual analysis of the overall effect and consequences of a large XACML policy set is a very daunting and time-consuming task.

In this thesis we address the problem of understanding the semantics of access control policy language XACML, in particular XACML version 3.0. The main focus of this thesis is *modelling and analysing access control policies in XACML 3.0*.

There are two main contributions in this thesis. First, we study and formalise XACML 3.0, in particular the Policy Decision Point (PDP). The concrete syntax of XACML is based on the XML format, while its standard semantics is described normatively using natural language. The use of English text in standardisation leads to the risk of misinterpretation and ambiguity. In order to avoid this drawback, we define an abstract syntax of XACML 3.0 and a formal XACML semantics. Second, we propose a logic-based XACML analysis framework using Answer Set Programming (ASP). With ASP we model an XACML PDP that loads XACML policies and evaluates XACML requests against these policies. The expressivity of ASP and the existence of efficient implementations of the answer set semantics provide the means for declarative specification and verification of properties of XACML policies.

Overall, we focus into two different area. The first part focuses on the access control language, more specifically is on the understanding XACML 3.0. The second part focuses on how we use Logic Programming (LP) to model access control policies. We show that there is a relation between XACML and LP through their semantics. We close the thesis by presenting applications in analysing access control properties and a case study. These applications show that these two approaches (AC paradigm and LP paradigm) can be combined together. We present access control security policies in a Smart Grid from Smart Meter perspective.

Keywords: Access Control Policies, IT Security, Control Systems, XACML, Composition Policies, Logic Programming, Answer Set Programming, Smart Grid, Smart Meter

Resumè

XACML (eXtensible Access Control Markup Language) er et fremtrædende adgangskontrol sprog, der er udbredt både i industrien og den akademiske verden. XACML er en international standard inden for informationssikkerhed. Problemet med XACML er, at dens specifikationer er beskrevet i naturligt sprog (jf [GM03, Mos05, Ris13]) og manuel analyse af den samlede effekt og konsekvenser af et stort XACML sæt politikker er en meget tidskrævende opgave.

I denne afhandling løse vi problemet med at forstå semantikken i adgangskontrolpolitik sproget XACML, især XACML version 3.0. Hovedfokus i denne afhandling er *modellering og analyse af adgangskontrol politikker i XACML 3,0.*

Der er to væsentlige bidrag i denne afhandling. Først, studerer og formaliserer vi XACML 3,0, især den Policy Decision Point (PDP). Den konkrete syntaks i XACML er baseret på XML-formatet, mens dens standard semantik beskrives normativt ved hjælp af naturligt sprog. Brugen af engelsk tekst i standardisering fører til risiko for fejlfortolkning og tvetydighed. For at undgå denne ulempe, definerer vi en abstrakt syntaks i XACML 3,0 og en formel XACML semantik. For det andet foreslår vi en logik-baseret XACML analyse, der anvender Answer Set Programming (ASP). Med ASP modellerer vi en XACML PDP, der indlæser XACML politikker og vurderer XACML anmodninger mod disse politikker. Ekspressiviteten af ASP og eksistensen af effektive implementeringer af *answer set semantics* giver midlerne til deklarativ specifikation og verifikation af egenskaber XACML politikker. Vi præsenterer adgangskontrol sikkerhedspolitikker i et Smart Grid fra Smart Meter perspektiv.

Nøgleord: Adgangskontrolspolitikker, it-sikkerhed, kontrolsystemer, XACML, Komposition Politikker, Logic Programming, Answer Set Programming, Smart Grid, Smart Meter

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science, the Technical University of Denmark in partial fulfilment of the requirements for acquiring the PhD degree in Computer Science.

The PhD study has been carried out under the supervision of Professor Hanne Riis Nielson and Professor Flemming Nielson in the period of September 2010 to March 2015 (excluding the maternity leave period from March 2013 to June 2014). The PhD project was partially supported by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology.

Most of the work behind this dissertation has been carried out independently and I take full responsibility for its contents. A part of the scientific work in this thesis is based on our published work in [RNN12,RNN13a,RNN13b,RNN14] with my two supervisors as co-authors. Another part is based on our published work in [VYR12] in collaboration with Roberto Vigo and Ender Yüksel (at that time all affiliated with the Technical University of Denmark).

Odense, March 2015

Carroline Dewi Puspa Kencana Ramli

Acknowledgements

First and foremost, I would like to express very special thanks to Martin Slota who has always been patient heartily supported me, listened to me and given good advice, in both my life and my study. To Alan Cahaya Ramli Slota, you are my son and sun, who has always been cheerful and made my days brighter. And to my parents for all their selfless care and support throughout my life. Without their encouragement, I would not have been able to finish my PhD.

I would like to express my deepest gratitude and appreciation to my supervisors, Hanne Riis Nielson and Flemming Nielson, for their continuous support and fruitful discussions. They have always been ready to provide priceless help and advice, and have given me freedom to explore and learn from my own mistakes.

Special thanks go to my old friends who helped me fighting this PhD: Arnold Wiliem, Monica Lestari Paramita, and to Ilham Winata Kurnia who always spent several time for proofreading my work,

Special thanks go to Ender Yüksel and Natalia Skrypnyuk for being my mentors in the early of my stay.

I would also like to thank Laus Brock-Nannestad for helping me with the Danish resumè in this thesis.

I would like to thank current and former members of the Language-Based Technology group at DTU: Alessandro, Ximeng, Marieta, Zaruhi, Omar, Lars, Laust, Erisa, Roberto, Jose, MichaĺI, Alejandro, Fuyuan, Piotr, Matthieu, Kebin, Michael, Christian, Alberto, Sebastian, Lijun. Thank you for being there

when I needed it the most

Finally, I would like to thank the secretaries as well as administrators at DTU. Special thanks go to Marian S. Adler, for caring a lot about all administrative and bureaucratic obstacles during my first time in DTU. To Lotte Skafte Jespersen who has always been ready to provide priceless help and advice in everyday matters. And to Ann-Cathrin Dunker who helped me with the bureaucratic staff at the end of my study.

Odense, March 2015

Carroline Dewi Puspa Kencana Ramli

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Introduction

To every problem, there is a most simple solution.

Agatha Christie, *The Clocks*

The thesis addresses the problem of understanding the semantics of access control policy language eXtensible Access Control Markup Language (XACML), in particular XACML version 3.0. The main focus of this thesis is

Modelling and analysing access control policies in XACML 3.0.

We first simplified the language by considering only the important parts in XACLM 3.0, and then we present the semantics of XACML 3.0 in a formal way. To make practical contribution, we present a systematic technique for transforming XACML 3.0 policies in logic programs, and then we demonstrate how the $\Pi_{\text{XACML}}-$ a program obtained by transforming XACML 3.0 policies into logic programs – makes it possible to use off-the-shelf answer set solvers to formally verify properties of access control policies represented in XACML 3.0.

The outline of this chapter is as follows. We first present motivation in Section 1.1. In Section 1.2 we show related work and problems that we might

handle through the thesis. Finally, we conclude the chapter by providing a brief summary of our contributions and guiding the reader to the road map of the thesis in Section 1.3.

1.1 Motivation

Our daily live is full of access control systems such as keys to doors, guest invitation lists, passports and visas for entering foreign countries, memberships in clubs. In the early computer era, access control systems were designed to mimic one or other of these "real world" systems. Capability access control [Lev84] resembles keys, i.e., the access to an object is granted whenever you have a capability for it. Identity-based capability [Gon89a, Gon89b] is a refinement of capability to incorporate subject identities. It is like a passport and visa, where an access to a country is given into particular subjects. Access control list [PP89] is like a fitness center membership – you get services based on your membership status.

When the usage of computer systems grew so fast and became more complex than not just simply to solve simple computation problems, the term *computer security* became necessary. A trusted computer system is a must. A fundamental mechanism to maintain security in computers systems is through controlling the accesses to the systems. Access control plays an important role in overall system security.

We need a formal language to model access control policies. We use the language to make the policies to be understandable in a unique way by anyone so that there is no ambiguous in the interpretation of the policies. We also need the language in the development of tools or software that can deal with such policies. Samarati and di Vimercati [SD01] defined the standard basic concepts behind access control policies. An access control policy language should

(i) ideally provide a small set of policy composition operators that are simple to understand and use,

(ii) expressive enough to capture all needed policy compositions,

(iii) flexible enough to compose the basic rules within a policy as well as whole policies,

(iv) be based on a mathematical foundation that provides for intellectual tractability and efficient analysis, and

(v) be intuitive and useful for normal developers.

There exist a number of policy languages such as

- XACML (eXtensible Access Control Markup Language) – an access control language based on XML [GM03, Mos05, Ris13],
- EPAL (Enterprise Privacy Authorization Language) – a security privacy language built by IBM [AHK+03, BMR04],
- Lithium – a policy language using a fragment of first-order logic [HW03, HW08],
- AspectKB – a formal language for specifying distributed systems with the ability of specifying security policies in each location based on aspect-oriented programming [HNN09].

Each of these languages has its own strengths and weaknesses. For example, XACML and EPAL are intuitive enough for the developers because they use XML, however their semantics are not formally defined. Thus it makes the behaviour of the languages not computationally tractable. XACML and EPAL fulfil the property number (v) but they fails to carry out property number (iv). On the other hand, Lithium and AspectKB are defined using mathematical formulation but they are not use mostly in industry.

This thesis focuses on exploring and designing an access control policy language which can be used in the industry and is also computationally tractable. Thus, we restrict our attention only two kinds of access control languages, i.e., XML-based access control language and logic-based access control language.

1.1.1 XML-Based Access Control Languages

XML (eXtensible Markup Language) [BPSM+08] is a markup language standardised by the World Wide Web Consortium (W3C). The purpose of XML is to describe data in a format that both human- and machine- understandable, thus, it can be used for information exchange on the Internet.

XML-based access control language takes the advantages of XML to address a language for expressing access control policies based on a high level formulation. The XML-based policies can be easily interchangeable through different systems using the same access control language. This feature is particularly interesting

in an open environment like the Internet, where a single system, which has to be protected as a single entity, may be distributed over the Net [dVFJS07].

Some relevant XML-based access control languages are WS-Policy [BBC+06a], the Enterprise Privacy Authorisation Language (EPAL) [BMR04, AHK+03], and the eXtensible Access Control Markup Language (XACML) [GM03, Mos05, Ris13].

WS-Policy. WS-Policy is a W3C recommendation (September 2007) for expressing Web service policies. WS-Policy includes:

- WS-PolicyAssertions [BHK+03]: a set of general messaging related assertions,
- WS-SecurityPolicy [DLGHB+05]: a set of security policy assertions related to WS-Security, and
- WS-PolicyAttachment [BBC+06b]: a specification how to add policies to Web services or other subjects such as service locators.

EPAL. EPAL (Enterprise Privacy Authorization Language) is an interoperability language for writing enterprise privacy policies to govern data handling practices in IT systems according to fine-grained enterprise privacy policies. An EPAL policy is essentially a set of privacy rules. A rule is a statement that includes a ruling, a data user, an action, a data category, and a purpose. A rule may also contain conditions and obligations. Each rule contains a precedence level. An EPAL policy document consists of three main sections:

- A policy identifier. It consists of information such as Issuer, Version Number, Start Date, End Date, Replacement Policy Name, Replacement Policy Version.
- Definitions of the possible components that can be used in the following rules. Here is where Data Users, Data Categories, Purposes, Actions, Context Models, Conditions and Obligations are defined.
- A rule that defines whether Data Users are allowed (`allow`) or denied (`deny`) to perform Action on Data Category for Purpose under Conditions.

XACML. XACML is an international standard in the field of information security. In January 2013, XACML version 3.0 was ratified by OASIS[1]. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies (*who can do what when*) while the request/response language expresses queries about whether a particular access should be allowed (*requests*) and describes answers to those queries (*responses*). XACML has been under development for some time. In this thesis we consider the newest version called XACML 3.0 [Ris13] which differs from previous versions in important ways.

XACML represents a shift from a more static security approach as exemplified by ACLs (Access Control Lists) towards a dynamic approach, based on Attribute Based Access Control (ABAC) systems. These dynamic security concepts are more difficult to understand, audit and interpret in real-world implications. The use of XACML requires not only the right tools but also well-founded concepts for policy creation and management.

The problem with XACML is that its specification is described in natural language (c.f. [GM03, Mos05, Ris13]) and manual analysis of the overall effect and consequences of a large XACML policy set is a very daunting and time-consuming task. How can a policy developer be certain that the represented policies capture all possible requests? Can they lead to conflicting decisions for some request? Do the policies satisfy all required properties? These complex problems cannot be solved easily without some automatised support. The thesis focuses on XACML since it is a prominent access control language that is widely adopted both in the industry and the academia.

1.1.2 Logic-Based Access Control Languages

Logic-based access control languages are particularly attractive as policy specification languages. One obvious advantage of logic-based languages is that their semantics is clear and unambiguous. The semantics is given formally so that the meaning of security policies stated using the language can be precisely determined. Second, logic-based language are expressive to represent access control policies without the extra problems that natural language would bring us. The declarative nature of logic-based languages yields a good compromise between expressiveness and simplicity. Their high level of abstraction is very close to the natural language formulation of the policies. Consequently, policy administrators model security policies easier and simpler using logic-based languages than

[1] The Organization for the Advancement of Structured Information Standards (OASIS) is a global consortium that drives the development, convergence, and adoption of e-business and web service standards.

using imperative programming languages.

The first work investigating logic languages for the specification of authorisations is the work by Woo and Lam [WL93]. They proposed a logical approach to representing and evaluating authorisation. The proposed language is designed to specify policy bases which encode sets of authorisation requirements. The precise semantics of policy bases is based on a formal notion of authorisation policy. The semantics is computable via a translation to extended logic programs.

Several of the most recent language designs rely on concepts and techniques from logic, specifically from logic programming [DeT02, Jim01, LGF03, LM03, LMW02, HW03, HW08].

1.2 Related Work

There are some approaches to defining access control policies in logic programs, for instance, Barker et al. use constraint logic program to define role-based access control in [BS03], while Jajodia et al. adopt the FAM / CAM language [JSSB97] – a logical language that uses a fixed set of predicates. However, their approaches are based on their own access control policy language whereas our approach is to define a well-known access control policy language, XACML.

There are two main differences of our work with other approaches. First, we consider the newest standard of XACML 3.0, ratified in January 2013, while most of other work considers XACML 2.0. XACML 2.0 differs XACML 3.0 not only in syntax but also in the treatment of extended indeterminate values. XACML 2.0 does not accommodate extended indeterminate values. Consequently, its combining operators are simpler than XACML 3.0 ones. Second, our work emphasises on defining the semantics of XACML in the first place. We focus on eliminating the ambiguities in the natural language specification of XACML.

Bryans explores the use of process algebra in formalising and analysing XACML [Bry05]. He presents the core concepts of XACML using Communicating Sequential Processes (CSP). Later, Bryans and Fizgerald present a formal semantics of XACML 2.0 in the formal specification language of Vienna Development Method (VDM++) [BF07]. The challenge occurs here because the semantics of XACML is presented in either CSP or VDM++. Thus, in order to understand the behaviour of XACML, first we should understand CSP or VDM++.

Halpern and Weissman [HW08] describe an XACML formalisation using First

Order Logic (FOL). However, their formalisation only captures a small fragment of the XACML specification. Kolovski et al. [KH07, KHP07, Kol08] provide a formalisation of XACML that explores the ground between propositional logic analysis tools (such as Margrave[2] [FKMT05, TK06]) and full first-order logic XACML analysis tools like Alloy [Jac02]. As a basis for the XACML formalisation, they use description logic (DL), which is a family of languages that are decidable subsets of FOL and are the basis for the Web Ontology Language (OWL). They map a large fragment of XACML into DL but they leave out the formalisation of the only-one-applicable combining operator, one of XACML combining operators.

Another approach is to represent XACML policies in terms of Answer Set Programming (ASP). Ahn et al. give a complete XACML formalisation in ASP [AHLM10a]. Ahn et al. translate the XACML specification directly into logic programming, so the ambiguities in the natural language specification of XACML are also reflected in their encoding.

While other approaches directly go to the implementation, Masi et al. [MPT12] define a formal semantics of XACML first and later, relying on their formal semantics of XACML, they build an implementation using Java and ANTLR tool for parsing generation. With this approach, they have the same intention as our work, viz. to clarify all ambiguous and intricate aspects of the XACML standard. Despite having the same goal their semantics is based on older version, XACML 2.0, while our work is based on XACML 3.0. Moreover, implementing XACML in procedural language such as Java has a drawback, i.e., it is hard to show that the results of the implementation agree with their semantics.

1.3 Contributions and Road Map

There are two main contributions in this thesis. First, we study and formalise XACML 3.0, in particular the Policy Decision Point (PDP). The concrete syntax of XACML is based on the XML format, while its standard semantics is described normatively using natural language. The use of English text in standardisation leads to the risk of misinterpretation and ambiguity. In order to avoid this drawback, we define an abstract syntax of XACML 3.0 and a formal XACML semantics. Second, we propose a logic-based XACML analysis framework using Answer Set Programming (ASP). With ASP we model an XACML PDP that loads XACML policies and evaluates XACML requests against these policies. The expressivity of ASP and the existence of efficient implementations

[2] `http://www.margrave-tool.org`

of the answer set semantics provide the means for declarative specification and verification of properties of XACML policies.

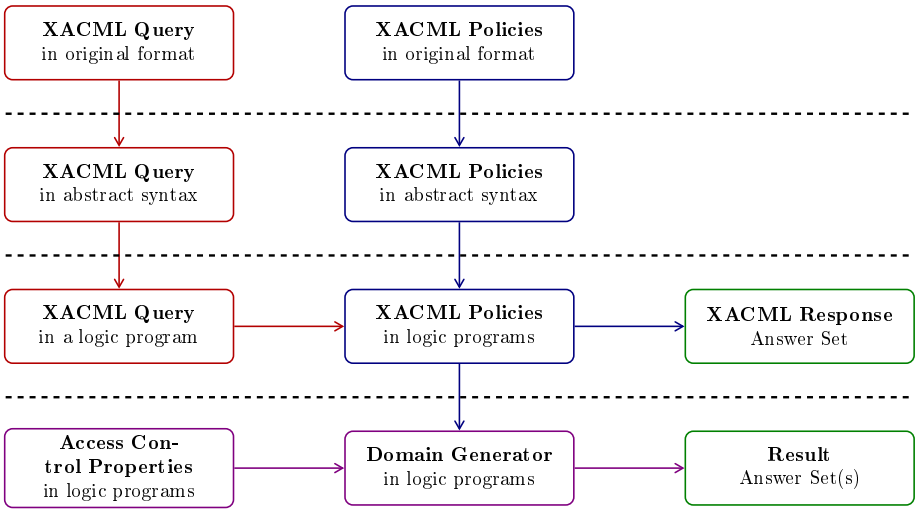

Figure 1.1: Summary of Our Work

Our work is depicted in Figure 1.1. There are two main modules, viz. the PDP simulation module and the access control (AC) security property verification module. The first module transforms policy files and queries from the original format in the XML syntax into an abstract syntax, which is more compact than the original format. Subsequently we generate a query program and an XACML policies program that correspond to the XACML query and the XACML policies, respectively. In the second module, we demonstrate how our results make it possible to use off-the-shelf Answer Set Programming (ASP) solvers to formally verify properties of AC policies represented in XACML.

Overall, we focus into two different area. The first part focuses on the access control language. More specifically our focus is on the understanding XACML 3.0. The second part focuses on how we use Logic Programming (LP) to model access control policies. We show that there is a relation between XACML and LP through their semantics. We close the thesis by presenting applications in analysing access control properties and a case study. These applications show that these two approaches (AC paradigm and LP paradigm) can be combined together.

This thesis contains five parts. The road map of this thesis is presented in Figure 1.2.

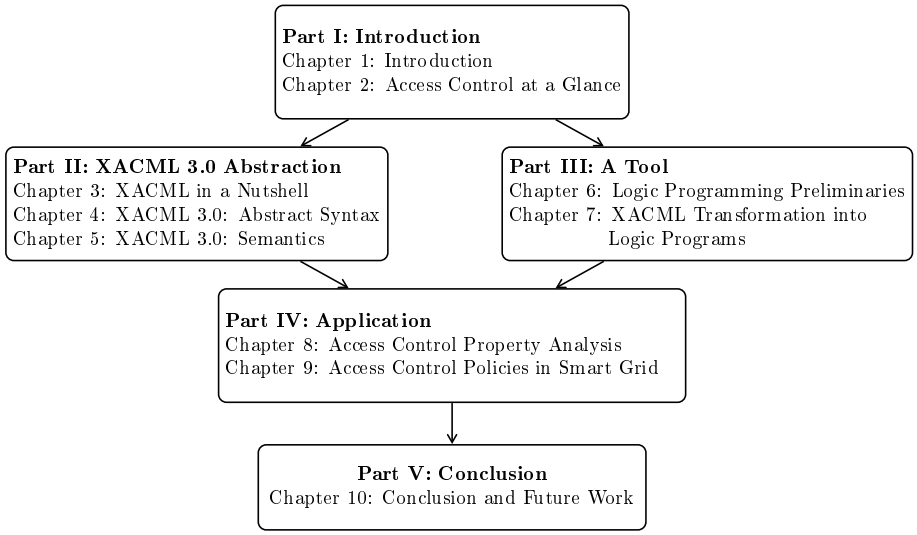

Figure 1.2: Thesis Road Map

Part I: Introduction. This is the introduction chapter and in Chapter 2 we present a review on access control system at a glance.

Part II: XACML 3.0 Abstraction. Part two discusses XACML abstraction. First of all, we discuss about the background of XACML in Chapter 3. After understanding the XACML model, in Chapter 4 we define a new syntax which is simpler and smaller than the original one. The purpose of this syntax is to capture only the important part of the XACML components. Armed with a smaller syntax, we are ready to define the semantics of XACML. In Chapter 5 we discusses all of XACML components semantics including combining operators – algorithms which are used in order to combine two or more access control policies. The work in this part is based on [RNN12, RNN14].

Part III: A Tool. After fully understanding XACML we are ready to design a tool that later we can use it for various analysis. The second part of the thesis focuses on building a tool based on XACML. We use the Logic Programming paradigm in our tool. In Chapter 6 we present preliminaries on Logic Programming. In Chapter 7 We show how XACML can be transformed into logic programs and we call Π_{XACML} for a program obtained by transforming XACML policies into logic programs and we show a semantic relation between XACML and Π_{XACML} in the same chapter. The work in this part is based

on [RNN13a, RNN13b].

Part IV: Application. The third part of the thesis focuses on the applications of Π_{XACML}. We start by doing analysis on access control properties. There are several combinatoric problems in analysis access control properties. In most cases, ASP solvers can solve combinatoric problems efficiently. In Chapter 8 we present a mechanism for verifying security properties against a set of access control policies using ASP tool. We show a mechanism to check whether a set of policies is complete (gap-free). In addition, we introduce a new semantics of XACML that may capture conflicted policies and later we present a mechanism on detecting conflict between access control policies. Last, we present an analysis on irrelevant policies. A policy is relevant if there is a request such that the decision is made based on this policy. Usually in a big set of policies, there is a policy that is irrelevant. This happens because policies are built based on several components and combined together. The last part of the thesis is a case study. We present access control security policies in a Smart Grid from Smart Meter perspective in Chapter 9. The work in this part is based on [RNN13a] and partly based on [VYR12].

Part V: Conclusion. The thesis ends with conclusion and future work in Chapter 10.

Chapter 2

Access Control at a Glance

> The leader said "Open Sesame!" and before Ali Baba's amazed eyes the sealed mouth of the cave magically opened and the men disappeared inside. To come out and close the entrance, the leader said "Close Sesame" and the cave sealed itself once more.
>
> Anonymous, translated by Richard Francis Burton
> *Ali Baba and The Forty Thieves*

In this chapter we present a review on access control systems. The chapter starts with an overview on how to design access control systems. In Section 2.2 we present three abstractions of controls that the designers of access control system should consider: access control policy, model, and mechanism. This thesis focuses on analysing and modelling access control policy. Thus, we emphasise on explaining its background. Generally, there are two kinds of access control policies, i.e., Discretionary Access Control (DAC) policies (Section 2.3) and Non-Discretionary Access Control (NDAC) policies (Section 2.4). For NDAC, we only cover the discussion on mandatory access control, role-based access control, and attribute-based access control, and not beyond that.

This chapter is based on Hu et al. [HFK06], Samarati and di Vimercati [SD01], and Gollmann [Gol11].

2.1 Overview

Access control is a mechanism to protect (tangible) asset in the system from unauthorised use by mediating how an access to the resource is granted or denied. Access control mechanisms concern only tangible assets such as treasure, buildings, documents, information, equipment, personnel, etc. The access control mechanisms exclude intangible asset such as efficiency, accuracy, performance, status, since they cannot be protected directly. Sterne stated that the term of unauthorised access should encompass two cases [Ste91]. The first case is a subject which *exceeds* its authorisation, for example, a personnel accesses classified document the sensitivity of which is higher than his or her clearance. The second case is a subject which *abuses* its authority, for example, a financial officer might initiate a fraudulent financial transaction.

The fundamental design of access control system consists of (see Figure 2.1 as an illustration):

- a *subject*: an active entity which initiates a query to access a resource,
- an *object* or a resource: a passive entity that needs protection from unauthorised use,
- an *action*: an operation that the subject does to the resource when the access is granted, and
- a *reference monitor*: an abstract machine that mediates all accesses to objects by subjects.

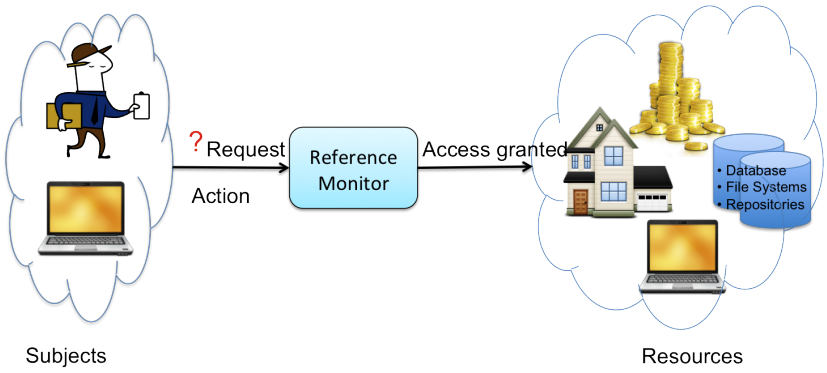

Figure 2.1: The Fundamental Model of Access Control.

The first step in developing access control system is to identify the *objects* that needs protection, the *subjects* which initiate the access requests to the objects and execute activities and the *actions* that can be executed on the objects and must be controlled [SD01].

2.2 Access Control Policies, Mechanisms, and Models

There are three abstractions of controls that the designers of access control system should consider: access control policy, model, and mechanism [HFK06].

Access control *policies* are high-level requirements that specify regulations on how to manage the access. Access control policies can be application-specific and thus the application owners should take the consideration. To capture the security requirements precisely, the designer of access control system should consider 5W+1H questions (Who, What, When, Where, Why, How) when designing access control policies:

- WHO can access,
- to WHAT resource,
- from WHAT device,
- WHEN the access is allowed (or denied),
- at WHAT time,
- WHERE the access is allowed (or denied),
- from WHAT location,
- WHY the access is allowed (or denied), for WHAT purpose, and
- HOW the access is given.

Access control *mechanism* describes how the system enforces access control policies. The access control mechanism must work as a reference monitor [U.S85]. Anderson defined a reference monitor concept as a set of design requirements on a reference validation mechanism, which enforces an access control policy over subjects' ability to perform operations on objects in a system [And72]. A reference monitor should have the following properties:

- tamper-proof: the reference validation mechanism should be temper proof.
- complete mediation: the reference validation mechanism must mediate all accesses to the system and its resources.
- verifiable: the reference validation mechanism must be small enough to be subject to analysis and tests, the verification methods and the completeness of which can be assured.

Access control *models* are the formal presentations of the security properties enforced by an access control system. Both users and application owners are interested on access control models. The models bridge the gap in abstraction between policy and mechanism. Using formal definitions, the application owners and system developers can define security properties precisely and unambiguously. Thus, it leads to the possibility of proving security properties. Therefore, we can argue that the system is "correct and secure" by proving that the model is "secure" and the mechanism "correctly implements" the models [Lan81].

2.3 Discretionary Access Control

The Discretionary Access Control (DAC) policies enforce access control based on the identity of the subject (requestor) and on explicit access rules stating what subjects are (or are not) allowed to do. The owner of the object or the person who has authorisation to control the object's access has the discretion to decide who can have access to the objects. In general, DAC is used to limit a user's access to an object [NSP94].

A simple access control model for DAC is the access control matrix. Lampson [Lam74] made the initial proposal to use the matrix for protecting resources in the context of operating systems. Graham and Denning [GD72] later refined the definition of the matrix for practical implementation. Harrison, Ruzzo, and Ullman [HRU76] formalised the Lampson's model for analysing the complexity of determining an access control policy.

In the access matrix model, the state of the system is defined by a triple (S, O, A), where S is the set of subjects, O is the set of objects, and A is the access matrix, where rows correspond to subjects, columns correspond to objects. An entry $A[s, o]$ reports the privileges of s on o. Figure 2.2 illustrates an example of an access matrix.

The decision policy – either to grant the access or to deny it – for access control matrix is based on the entry where the subject row meets the object column.

	Code 1	**Code 2**	**File 1**	**File 2**
Ali	read, write, execute	-	read	-
Baba	-	read, write, execute	-	read
Cashim	read	read	read, write	read, write
Mustafa	execute	execute	-	-

Figure 2.2: Access Control Matrix

For example, `Ali` may `read` `Code 1` and `File 1`, but not the case for `Baba`, since there is no entry in column `Code 1` and `File 1` in row `Baba`.

It is too complex to manipulate the matrix directly because the number of objects can be very large. Also a matrix for a real system tends to be very sparse, so most systems do not store the access rights in a matrix form. Rather, they use either the Access Control List (ACL) approach or the Capability approach. In the ACL approach, the matrix is viewed by column (see Figure 2.3 as illustration). Each object is associated with an ACL which stores the subjects and their access rights for the objects. The list is checked to see whether to grand an access. In the Capability approach, the matrix is viewed by row (see Figure 2.4 as illustration). Each subject is associated with a Capability list which stores its access rights to all concerned objects. If it is no in the table, the subject does not have any access to the object. Possessing a capability is the proof of possessing the corresponding access rights.

	Code 1
Ali	read, write, execute
Cashim	read
Mustafa	execute

	File 1
Ali	read
Cashim	read, write

	Code 2
Baba	read, write, execute
Cashim	read
Mustafa	execute

	File 2
Baba	read
Cashim	read, write

Figure 2.3: Access Control Lists

The drawbacks of DAC are as follows [HFK06]:

- The privileges for accessing objects are decided by the owner of the object, rather than through a system-wide policy that reflects the organisation's security requirements.
- There is no assurance on the information flow in the system since information can be copied from one object to another.

	Ali
Code 1	read, write, execute
File 1	read

	Baba
Code 2	read, write, execute
File 2	read

	Cashim
Code 1	read
Code 2	read
File 1	read, write
File 2	read, write

	Mustafa
Code 1	execute
Code 2	execute

Figure 2.4: Capabilities

– There is no restrictions apply to the usage of information when the subject has already connected to the system. Thus, DAC policy is vulnerable to Trojan Horse attacks. A Trojan Horse is a computer program with an apparently or actually useful function, but contains additional hidden functions that surreptitiously exploit the legitimate authorisations of the invoking process.

2.4 Non-Discretionary Access Control

Non-Discretionary Access Control (NDAC) is a category for all access control policies other than DAC. There are several NDAC policies such as Mandatory Access Control (MAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC).

2.4.1 Mandatory Access Control

Mandatory Access Control (MAC) is the most mentioned NDAC policy. The *Mandatory Access Control* policies enforce access control based on regulations made by a central authority. Contrary to DAC, in MAC the owner of an object has no right to change an authorisation to the access. MAC implementations mostly take place in military security [PP89] where an individual data owner does not decide who has a Top Secret clearance nor can the owner change the classification of an object from Top Secret to Secret. The objects might be assigned to hierarchically security level according to National Security Information:

- Top Secret (TS): shall be applied to information, the unauthorised disclosure of which reasonably could be expected to cause exceptionally grave damage to the national security.
- Secret (S): shall be applied to information, the unauthorised disclosure of which reasonably could be expected to cause exceptionally serious damage to the national security.
- Confidential (C): shall be applied to information, the unauthorised disclosure of which reasonably could be expected to cause exceptionally damage to the national security.
- Unclassified (U): no security restriction.

Classification is the security level given to information based on policy, We use the same classification levels used for clearances to assign classifications. Clearance and classification go together, in that, a user's clearance is a limit to the access of information based on the information's classification. Classification relation is $\mathsf{TS} > \mathsf{S} > \mathsf{C} > \mathsf{U} > \mathsf{null}$. Each security level is said to dominate itself and all others below it in this hierarchy.

The classification assigned to objects and subjects in multilevel MAC depends on whether it is intended for confidentiality or integrity. A confidential mandatory policy controls the direct and indirect flows of information to prevent leakages to unauthorised subjects. Bell and LaPadula [BPCF73, BLD73] formulated two principles that a system should meet to protect information confidentiality:

no-read-up A subject is allowed a `read` access to an object only if the access class of the subject dominates the access class of the object.

no-write-down A subject is allowed a `write` access to an object only if the access class of the subject is dominated by the access class of the object.

In the Bell and LaPadula model, a system is composed of sets of subjects S, objects O, actions A, and security levels L with a partial ordering $\geq$. A configuration (state) $v \in V$ is defined as triple (b, M, f) where:

- $b \in B = \mathcal{P}(S \times O \times A)$ as a set of triples (s, o, a) indicating that the subject s is currently performing an operation a on the object o.
- $M \in \mathcal{M} = (S \times O \to \mathcal{P}(A))$ is an access control permission matrix.
- $f = (f_S, f_C, f_O) \in F = (S \to L) \times (S \to L) \times (O \to L)$ yields the maximal security levels of subjects (clearance), their possibly lower current level, and security levels of objects.

The Bell and LaPadula principles stated above are formulated as follows:

simple-property A state v satisfies the simple security property iff for every $s \in S, o \in O : (s, o, read) \in b \Rightarrow f_S(s) \geq f_O(o)$. This property captures no-read-up principle that applies when a person request access to a classified document.

***-property** A state v satisfies the *-security property iff for every $s \in S, o \in O : (s, o, write) \in b \Rightarrow f_O(o) \geq f_C(s)$.

The mandatory access control policy only protects the confidentiality of the information. However, there is no control enforced on its integrity. Low classified subjects could still be able to enforce improper indirect modifications to objects they cannot write. For instance, integrity could be compromised if the Trojan Horse implanted by a low classified subject in the application would write or modify data in a targeted file.

Consider the following example. Assume that within an organization, Vicky, a top-level manager, created a file `Recipe` containing important information about recipes of new products. This information is very sensitive for the organization and, according to the organization's policy, should not be disclosed to anybody besides Vicky. Consider now John, one of Vicky's subordinates. Possible classifications reflecting the access restrictions to be enforced could be: Secret for Vicky and `Recipe`, and Unclassified for John. Suppose John is also an infiltrator who is sent by a competitor organization. John wants to modify the Recipe so that it will give bad impact to the Vicky's organization. To achieve this, John modified an application generally used by Vicky, to include two hidden operations, read and write operations on file `Recipe`. Then, he gives the new application to his manager. Suppose now that Vicky executes the application. Since the application executes on behalf of Vicky, every access is checked against Vicky's authorisations with Secret as its security level. The read and write operations are allowed since Vicky's security level dominates `Recipe` security level. Thus, John can modify file `Recipe` without Vicky's notice.

Integrity-based mandatory policies enforce controls on the integrity of information which control the flow of information. The same as confidentiality, each subject and object in the systems is assigned an integrity classification. An example of integrity levels can be Crucial (`C`), Important (`I`), and Unknown (`U`). Biba [Bib77] proposed a dual policy for safeguarding the integrity of information. The semantics of integrity classifications is as follows:

- The integrity level associated with a user reflects the user's trustworthiness for inserting, modifying, or deleting information.

– The integrity level associated with an object reflects the degree of trust that can be placed on the information stored in the object (and also the potential damage that could result from unauthorised modifications of the information).

The formalisation of the Biba model is as follows. Consider integrity policies that label subjects and objects with elements from a lattice $(L, \geq)$ of integrity levels. For example $\texttt{C} \geq \texttt{I} \geq \texttt{U}$. The assignment of integrity levels to subjects (il_S) and objects (il_O) is given by the functions:

– $il_S : S \rightarrow L$, and

– $il_O : O \rightarrow L$.

Access control is enforced according to the following two principles:

simple integrity property A subject s is allowed a `write` access to an object o only if the access class of the subject dominates the access of the object, i.e., $il_S(s) \geq il_O(o)$. This property is also called *no-write(to)-up*.

integrity *-property A subject s is allowed a `read` access to an object o only if the access class of the object dominates the access of the subject, i.e., $il_O(o) \geq il_S(s)$. This property is also called *no-read(from)-down*.

Satisfying these principles safeguards the integrity by preventing information stored in low objects from flowing to higher, or incomparable objects. The integrity-based mandatory policies prevent subjects to indirectly modify information they cannot write. Continuing from previous example, possible integrity classification: Crucial for `Recipe`, Important for Vicky and `Recipe`, and Unknown for John. Hence, Vicky only can read `Recipe` but she cannot write or modify `Recipe`. However John can read `Recipe`, which is bad for the organization since `Recipe` is a secret. If both secrecy and integrity have to be controlled, objects and subjects have to be assigned two access classes, one for secrecy control and one for integrity control.

MAC policies guarantee better security than DAC policies since they can also control indirect information flows. However, mandatory policies may become too rigid. Several proposals attempt to combine mandatory flow control and discretionary authorisations, such as: the Chinese Wall policy [BN89] and authorisation-based information flow policies [Den76].

2.4.2 Role-Based Access Control

Ferraiolo and Kuhn [FK92] have formalised Role-Based Access Control (RBAC) as an alternative to the traditional DAC and MAC policies. The main basis of RBAC is to simplify administration of permissions for large numbers of users. It is more important to know what a user's organisational responsibilities are rather than who the user is. RBAC models categorise users based on similar needs and group them into roles. Permissions are assigned to roles rather than to individual users. RBAC merges the flexibility of explicit authorisations with the imposed organisational constraints.

RBAC contains roles, procedures, and possibly data types as intermediate layers between subjects and objects [FK92]. *Role* is a collection of procedures. Roles are assigned to users. A user assigned to a role can execute the procedures defined for that role. *Procedures* are "high-level" access control methods with a more complex semantics than just read or write. Procedures can only be applied to objects of certain data types. Each object is of a certain *data type* and can be accessed only through the procedures defined for this data type.

There are several benefits of RBAC, such as authorisation management, hierarchical roles, least privilege, separation of duties, and constraints enforcement. From the authorisation management perspective, role-based policies benefit from a logical independence in specifying user authorisations by breaking this task into two parts: the assignment of roles to users, and the assignment of authorisations to access objects to roles. This greatly simplifies the management of the security policy. For example when a new user joins the organisations, the administrator only needs to grant her the roles corresponding to her job; when a user's job changes, the administrator simply has to change the roles associated with that user; when a new application or task is added to the system, the administrator needs only to decide which roles are permitted to execute it.

Many applications naturally have a hierarchy of roles, based on the familiar principles of generalisation and specialisation. The role hierarchy can be exploited for authorisation inference. For instance, authorisations granted to roles can be propagated to their specialisations (e.g., the `secretary` role can be allowed all accesses granted to `adm_staff`). Authorisation inference can be enforced on role assignments, by allowing users to activate all generalisations of the roles assigned to them (e.g., a user of the role `secretary` will also be allowed to play the role `adm_staff`). Authorisation inference has the advantage of further simplifying the authorisation management.

Roles allow a user to sign on with the least privilege required for the particular task he/she needs to perform. Users authorised to powerful roles do not need to

exercise them until those privileges are actually needed. This minimises the risk of damage due to inadvertent errors, Trojan Horses, or intruders masquerading as legitimate users.

The separation-of-duties refer to the principle that no user should be given enough privileges to misuse the system on their own. For instance, the person authorising a paycheck should not be the same person who prepares them. Separation-of-duties can be enforced either statically or dynamically. In static separation-of-duties policies, the roles that may be assigned to a user are fixed and have to take into account separation-of-duties requirements. In dynamic separation-of-duties policies, the roles that may be assigned to a user depend on the current task.

Roles provide a basis for the specification and the enforcement of further protection requirements that real world policies may need to express. For instance, cardinality constraints can be specified, to restrict the number of users allowed to activate a role or the number of roles allowed to exercise a given privilege. The constraint can also be dynamic, i.e., be imposed on role activation rather than on role assignment. For instance, while several users may be allowed to activate role `chair`, a further constraint can require that at most one user at a time can activate it. The National Institute of Standards and Technology (NIST) [FSG+01] has published a widely used classification of RBAC levels. The levels are defined incrementally. Each level includes the features of the previous level.

- Flat RBAC: users are assigned to roles, permissions are assigned to roles, users get permissions via role membership; user-role reviews are supported.
- Hierarchical RBAC: adds support for role hierarchies.
- Constrained RBAC: adds support for separation-of-duties policies.
- Symmetric RBAC: adds support for permission-role reviews.

There are some issues missing from RBAC. The simple hierarchical relationship may not be sufficient to model the different kinds of relationships that can occur. Different ways of propagating privileges (delegation) should then be supported. For example, a secretary may need to be allowed to write specific documents *on behalf* of her manager, but neither role is a specialisation of the other.

The traditional concept of ownership may not apply any more: a user does not necessarily own the objects she created when in a given role. User's identities should not be forgotten. For example, a doctor may be allowed to specify treatments and access files but she may be restricted to treatments and files for

her own patients, where the doctor-patient relationship is defined based on their identities.

2.4.3 Attribute-Based Access Control

Traditional access control systems were based on the identity of the party requesting a resource. This mechanism is not effective in open environment such as the Internet. A more appropriate approach is where the access decision is based on properties (attributes) of the requester and of the resource. Attribute-Based Access Control (ABAC) is a "next generation" authorisation model that provide dynamic, context-aware and risk-intelligent access control. ABAC differs from the traditional discretionary access control model by replacing the subject by a set of attributes, and replacing the objects by descriptions in terms of available properties associated with them. The basic idea is that not all access control decisions are identity-based but rather based on digital credentials which are more suitable for the open communication infrastructure.

A subject is an entity (e.g., a user, application, or process) that takes action on a resource. Each subject has associated attributes which define the identity and characteristics of the subject. Such attributes may include the subject's identifier, name, organisation, job title, and so on. A subject's role, naturally, can also be viewed as an attribute. A resource is an entity (e.g., a web service, data structure, or system component) that is acted upon by a subject. As with subjects, resources have attributes that can be leveraged to make access control decisions. Resource attributes can often be extracted from the "metadata" of the resource. In particular, a variety of web service metadata attributes may be relevant for access control purposes, such as ownership, service taxonomy, or even Quality of Service (QoS) attributes. Environment attributes describe the operational, technical, and even situational environment or context in which the information access occurs. For example, attributes such as current date and time, the current virus / hacker activities, and the network's security level (e.g., Internet vs. Intranet), are not associated with a particular subject nor a resource, but may nonetheless be relevant in applying an access control policy.

Attributes are often retrieved from different information systems within the infrastructure. A policy can thus combine the state of data in many systems to resolve an authorisation request. In the example, the user's department may be retrieved from an HR system, the region from a CRM system and the document type from a document management system. Authorisation thereby enables integration to support workflows that incorporate IT support from multiple IT systems, something that is virtually impossible to handle with traditional access control models.

The main elements of the policy rules are the following.

- *Subject.* Each expression identifies a set of subjects having specific properties. Each user is then associated with a profile that defines names and values of some properties that characterise the user.
- *Object.* The characterisation of the entities to be protected should be specified through expressions. As for subjects, each object is associated with a profile which defines the names and values of their properties.
- *Actions.* Policies must be able to make distinctions based on the type of actions being requested on objects.
- *Purposes.* Data access requests are made for a specific purpose, which represents how the data is going to be used by the recipient.
- *Conditions.* Additional conditions such as conditions dictated by legislation, location-based conditions, and trust conditions.
- *Obligations.* To improve privacy, users can define some obligations attached to the data. Therefore, when a certain access is allowed, the parties involved must take some additional steps, following the defined obligations.

Each access request results in an access decision that can take three different forms:

- *yes*: the access request is granted,
- *no*: the access request is denied, and
- *undefined*: current information is insufficient to determine whether the request can be granted or denied.

Part II

XACML 3.0 Abstraction

Chapter 3

XACML 3.0 in A Nutshell

> When it comes to developing standards, you're either at the table menu ... or you're on the menu.
>
> OASIS

XACML (eXtensible Access Control Markup Language) is an approved OASIS[1] Standard access control language [GM03,Mos05,Ris13]. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies (*who can do what when*) while the request/response language expresses queries about whether a particular access should be allowed (*requests*) and describes answers to those queries (*responses*). XACML has been under development for some time – in this thesis we consider the newest version called XACML 3.0 [Ris13].

In this chapter we present a summary of XACML 3.0 as described in [Ris13]. For convenience, we use XACML as a reference to XACML 3.0 if nothing stated otherwise. In Section 3.1 we introduce XACML model. Then, in Section 3.2 we explain components of XACML policies and XACML combining algorithms. Finally, we explain algorithms used to combine XACML policies in Section 3.3.

[1] OASIS (Organization for the Advancement of Structured Information Standard) is a non-for-profit, global consortium that drives the development, convergence, and adoption of e-business standards. Information about OASIS can be found at `http://www.oasis-open.org`.

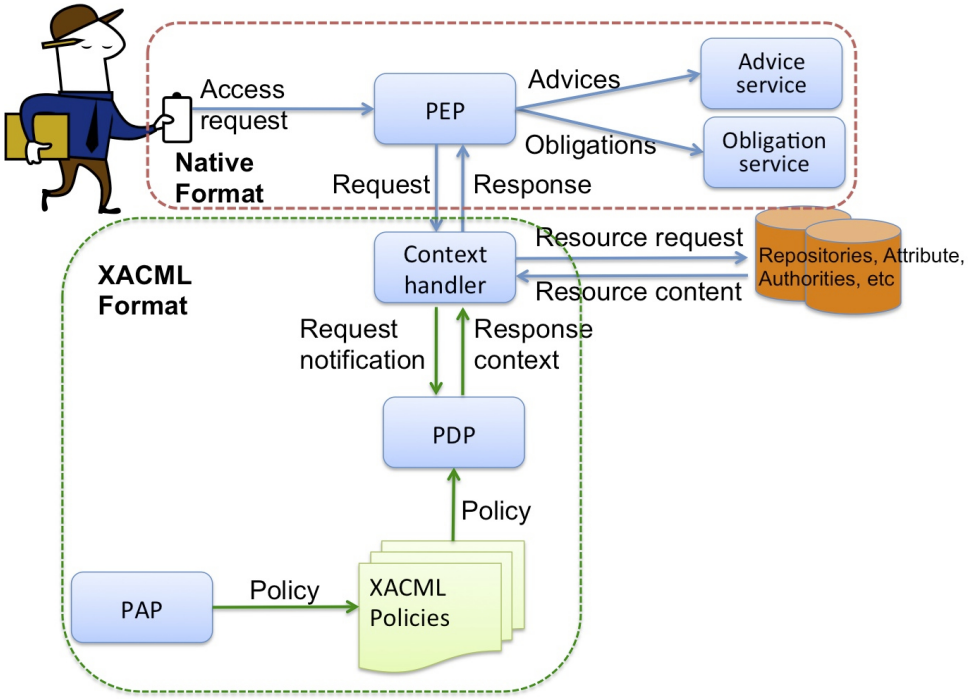

Figure 3.1: The basic model of XACML.

3.1 The XACML Model

A typical XACML usage scenario is illustrated in Figure 3.1. Here a subject (e.g. a human user or a program) wants to take some action on a particular resource. The subject submits its query to the entity protecting the resource (e.g. a file system or a Web server); this entity is called a *Policy Enforcement Point* (*PEP*). A PEP can come in many forms. For instance, a PEP may be a part of a remote-access gateway, a part of a Web server or a part of an email user-agent, etc. Multiple PEPs may have to enforce a particular policy. It would be inefficient to force a policy writer to write the same policy in several different ways in order to accommodate the format requirements for each PEP. Thus, there is a need for a canonical form of the request. The *Context Handler* takes care of the canonical XACML policy form.

The PEP forms a request to the Context Handler in its native request format, optionally including attributes of the subject, resource, action, environment

and other categories. Later, the Context Handler constructs a request (using the XACML request language) based on the attributes of the subject, action, resource, and other relevant information and sends it to the *Policy Decision Point* (*PDP*). Optionally, the Context Handler includes the resource in the context.

The *Policy Administration Point* (*PAP*) is in charge of writing access control policies in XACML policy language and make them available to the PDP. The PDP obtains the appropriate policies from the PAP and then it examines the request, retrieves policies that are applicable to this request, and determines whether or not access should be granted according to the XACML rules for evaluating policies. The PDP returns a decision with one of these values:

- Permit: the requested access is granted.
- Deny: the requested access is denied.
- Indeterminate: the PDP cannot determine whether the requested access is granted or denied because there is a computation error.
- NotApplicable: the PDP cannot find an applicable policy for the requested access.

The answer (expressed in the XACML response language) is returned to the Context Handler, which can then allow or deny the access of the requester, by translating the response context to the native response format into the PEP. The response may be occupied by obligations that are mandatory to be fulfilled, or by advice that may be bypassed. There are three schemes in PEP, namely base PEP, deny-biased PEP and permit-biased PEP.

Base PEP

- If the decision is Permit, then the PEP shall permit access. If obligations accompany the decision, then the PEP shall permit access only if it understands and it can and will discharge those obligations.
- If the decision is Deny, then the PEP shall deny access. If obligations accompany the decision, then the PEP shall deny access only if it understands and it can and will discharge those obligations.
- If the decision is NotApplicable, then the PEP's behaviour is undefined.
- If the decision is Indeterminate, then the PEP's behaviour is undefined.

Deny-biased PEP

- If the decision is Permit, then the PEP shall permit access. If obligations accompany the decision, then the PEP shall permit access only if it understands and it can and will discharge those obligations.
- All other decisions shall return in the denial of access.

Permit-biased PEP

- If the decision is Deny, then the PEP shall deny access. If obligations accompany the decision, then the PEP shall deny access only if it understands and it can and will discharge those obligations.
- All other decisions shall return in the permission of access.

3.2 The XACML Components

In order to obtain modularity in access control, XACML organises policies into several components. There are three levels of policies in XACML, namely PolicySet, Policy and Rule[2]. PolicySet or Policy can act as the root of a set of access control policies. Throughout this thesis we consider that PolicySet is the root of the set of access control policies.

A PolicySet is a collection of other PolicySets or Policies whereas a Policy consists of one or more Rules. A Rule is the smallest component of an XACML policy. Each Rule contains a Boolean expression which shows that the Rule grants an access (denoted by Permit) or denies an access (denoted by Deny).

As an illustration (see Figure 3.2), suppose we consider access control policies used within a National Health Care System. The system is composed of access control policies for the individual local hospitals. Each local hospital has its own policies such as patient policy, doctor policy, administration policy, etc. Each policy contains one or more rules, for example, in the patient policy there may be a rule saying that only the designated patient can read his or her record. In this illustration, both the National Health Care System and local hospital

[2]Please note that Policy and Rule written with an initial capital letter are intended for XACML components whereas policy and rule written in small letters are used as common English terminologies.

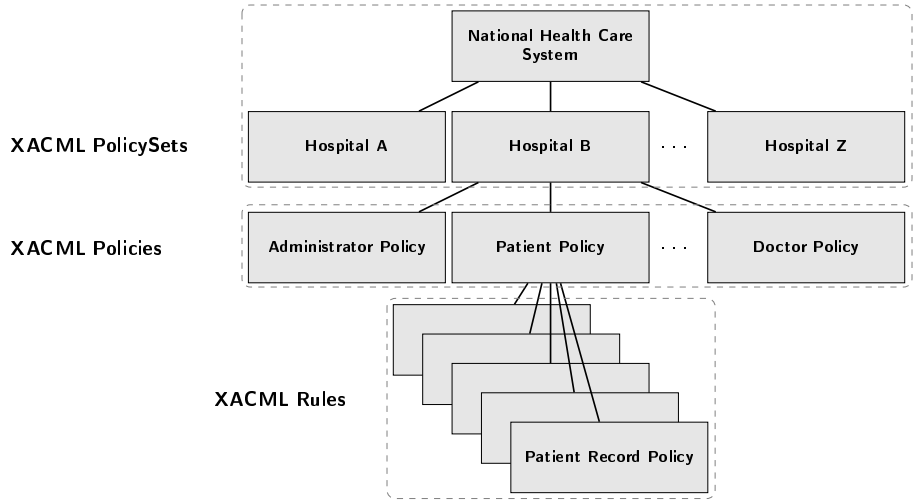

Figure 3.2: An example of Access Control Policies for a National Health Care System.

policies are PolicySet elements. However the patient policy is a Policy and one of its rules is the patient record policy.

Every policy is only applicable to a certain target. The Rule's Target defines the set of requests to which the Rule is intended to apply in the form of a logical expression on attributes in the request. The Target element may be absent from a Rule. In this case, the target of the Rule is the same as the parent Policy element. The policy writer may declare specifically the Target of the PolicySet (or Policy). Another option is that the Target of the PolicySet (or Policy) – we call it as the *outer component* – may be calculated from the Target elements of the PolicySet, Policy and Rule elements that it contains – we call it as the *inner components*. XACML uses two methods to calculate Target element:

- Union evaluation. In this case, the Target of the outer component is applicable if there is at least one of the Target element in the inner component matches to the decision request.
- Intersection evaluation. In this case, the Target of the outer component is applicable if every Target in the inner component matches to the decision request.

A policy is applicable when a request matches its target; otherwise, it is not applicable. The evaluation of composite policies is based on a particular *combining*

operator that combines decisions from multiple policies.

3.3 The XACML Policy Combining Algorithms

XACML defines a number of combining algorithms to define a procedure for arriving at an authorisation decision given the individual results of evaluation of a set of policies. Currently, XACML has twelve standard combining algorithms:

- (unordered- and ordered-) deny-overrides,
- (unordered- and ordered-) permit-overrides,
- deny-unless-permit,
- permit-unless-deny,
- first-applicable,
- only-one-applicable,
- legacy (unordered- and ordered-) deny-overrides, and
- legacy (unordered- and ordered-) permit-overrides.

The behaviour of the ordered combining operators are identical to the unordered combining operators with one exception, i.e., the order in which the collection of Rules (or Policies) is evaluated shall match the order as listed in the policy. The ordered combining operators might give different result with the unordered combining algorithms when the Obligations take place.

In the case of deny-overrides algorithm, if a single Rule or Policy is evaluated to Deny, then the combined result is Deny regardless of the evaluation result of the other Rule or Policy elements in the applicable policy. Likewise, in the case of permit-overrides algorithm, the combined result is Permit if there is a single Rule or Policy is evaluated to Permit. The legacy combining algorithms are similar to the non legacy ones. The difference is in the treatment of the Indeterminate value – which we will discuss the XACML decision values more deeper in the Semantics of XACML in Section 5.

In the case of the deny-unless-permit combining algorithm, if there is a single Rule, Policy, or PolicySet is evaluated to Permit, then the combined result is Permit. Otherwise, the result is Deny. The permit-unless-deny combining algorithm is the dual of the deny-unless-permit combining algorithm. In this case,

the Deny takes precedence. The combined result is Deny whenever there is a single Rule, Policy, or PolicySet is evaluated to Deny. Otherwise, it is Permit.

In the case of the first-applicable combining algorithm, the combined result is the same as the result of evaluating the first Rule, Policy or PolicySet element in the list of policy which Target is applicable.

The only-one-applicable combining algorithm only applies to set of policies. The result of this combining algorithm ensures that one and only one Policy or PolicySet is applicable by virtue of its Target. If no Policy or PolicySet applies, then the result is NotApplicable. If more than one Policies or PolicySets are applicable, the the result is Indeterminate. When exactly one Policy or PolicySet is applicable, then the result of the combining algorithm is the result of evaluating the single applicable Policy or PolicySet.

Chapter 4

XACML 3.0 Abstract Syntax

> If the syntax is good enough for the information, it should be good enough for the meta-information.
>
> Erik Naggum, a Norwegian computer programmer

The syntax of XACML is described verbosely in the XACML 3.0 specification [Ris13] since XACML 3.0 uses XML language to describe its syntax. We can see below an example of XACML 3.0 policies.

Example 4.1 (Taken from [Ris13] Section 4.1.1) Suppose that a corporation named Medi Corp (identified by its domain name: `med.example.com` has an access control policy that states (in English):

> Any user with an e-mail name in the "med.example.com" namespace is allowed to perform any action on any resource.

The XACML policy for this example is as follows.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Policy
3    xmlns="urn:oasis:name:tc:xacml:3.0:core:wd-17"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:
         wd-17
6    http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd
         -17.xsd"
7    PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1"
8    Version="1.0"
9    RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-
         overrides">
10   <Description>
11       Medi Corp access control policy
12   </Description>
13   <Target/>
14   <Rule
15     RuleId= "urn:oasis:names:tc:xacml:3.0:example:SimpleRule1"
16     Effect="Permit">
17     <Description>
18         Any subject with an e-mail name in the med.example.com
       domain can perform any action on any resource.
19     </Description>
20     <Target>
21       <AnyOf>
22         <AllOf>
23             <Match
24               MatchId="urn:oasis:names:tc:xacml:1.0:function:
       rfc822Name-match">
25               <AttributeValue DataType="http://www.w3.org/2001/
                     XMLSchema#string">med.example.com</
                     AttributeValue>
26               <AttributeDesignator
27                   MustBePresent="false"
28                   Category="urn:oasis:names:tc:xacml:1.0:subject
                         -category:access-subject"
29                   AttributeId="urn:oasis:names:tc:xacml:1.0:
                         subject:subject-id"
30                   DataType="urn:oasis:names:tc:xacml:1.0:data-
                         type:rfc822Name"/>
31             </Match>
32         </AllOf>
33       </AnyOf>
34     </Target>
35   </Rule>
36 </Policy>
```

The explanation of the code is as follows. (The number in front of the text indicating the line number(s)).

1 It is a standard XML document tag indicating which XML version is being

used and what the character encoding is.

2 – 9 `<Policy>` is a tag to start XACML Policy. It consists of XML namespace declaration (line 3 – 4), URN for the XACML policy schema (line 5 – 6), policy identifier (line 7), version control (line 8), and rule combining algorithm identifier (line 9).

10 – 12 Description of the policy.

13 Description of the Target element of this policy. In this case, the Target is empty.

14 – 38 The Rule element of this policy. It consists of Rule identifier (line 15) and its decision if the Rule is applicable (line 16). The Target of this Rule is described in lines 21 – 37.

There are several attributes and expressions that are not significantly describe XACML. In order to allow a succinct treatment of the important aspects of XACML we first develop an abstract syntax in this chapter. We focus only on the core aspects of XACML.

To summarise, some aspects that are not considered in this thesis are:

- Obligations and Advices. Since Obligations and Advices are not mandatory elements in XACML, and their actions do not play any role in the evaluation procedure, thus, for simplicity we do not consider Obligations and Advices in this thesis.
- AttributeDesignator and AttributeSelector. AttributeDesignator and AttributeSelector elements are used to identify one or more attribute values in the request context, which comes from Context Handler. In this thesis, we do not directly model these XACML elements, instead we assume the attribute values are already obtained from their respective sources.
- Apply. The Apply element denotes application of a function to its argument. For the simplicity, in this thesis we do not consider Apply element, instead the user freely to define any functions without any restriction that the function should be encapsulated in Apply element.
- MultiRequests. We do not consider MultiRequests in this thesis since it is optional. The MultiRequests element can be modelled by having Request element repeatedly.
- Version. The Version entity is mandatory in XACML 3.0. However, in this thesis we do not consider version control. Our main focus is to model XACML policies, thus, we omit this entity.

To make the notation clear we use:

- **bold** font to denote non-terminal symbols,
- `typewriter` font to denote terminal symbols,
- *italic* font to denote identifiers and values, and
- *calligraphic* font to denote XACML components.

We use the star symbol (*) to indicate that there is zero or more of the preceding element and we use the plus symbol ($^+$) to indicate that there is one or more of the preceding element. In XACML it is required that each policy must have a unique identifier (*id*) but this is not captured by the abstract syntax.

We present in Table 4.1 a summary of XACML 3.0 abstract syntax that is faithful to the more verbose syntax used in the standard [Ris13].

This chapter consists of three sections. In Section 4.1, we present the abstract syntax for XACML policies components. We continue by presenting the abstract syntax for XACML request component in Section 4.2. Finally, we present the abstract syntax for XACML response component in Section 4.3. In each section, we also present the original syntax of XACML 3.0 in order to show which aspects that we take and which one that we omit.

4.1 Abstract Syntax for XACML Policy Components

There are three main policy components in XACML, namely PolicySet, Policy and Rule. PolicySets and Policies are containers for other policies while a Rule is the most elementary unit of policy. The root of all XACML policies can be a PolicySet or a Policy.

4.1.1 PolicySet Element

A PolicySet is a top-level element in the XACML policy schema. The syntax of PolicySet is defined in [Ris13] as follows.

Table 4.1: Abstract syntax of XACML 3.0.

	XACML Policy Components	
PolicySet	::=	$\texttt{policyset}(id) : \{\, \mathbf{comb}_{id}\ ;\ \mathbf{Target}\ ;\ \langle (\mathcal{PS}_{id} \mid \mathcal{P}_{id})^* \rangle \,\}$
Policy	::=	$\texttt{policy}(id) : \{\, \mathbf{comb}_{id}\ ;\ \mathbf{Target}\ ;\ \langle \mathcal{R}_{id}^{+} \rangle \,\}$
Rule	::=	$\texttt{rule}(id) : \{\, \mathbf{Effect}\ ;\ \mathbf{Target}\ ;\ \mathbf{Condition} \,\}$
Target	::=	$\texttt{target}(id) : \{\, \mathsf{null} \,\}$
	|	$\texttt{target}(id) : \{\, \mathbf{AnyOf}^{+} \,\}$
AnyOf	::=	$\texttt{anyof}(id) : \{\, \mathbf{AllOf}^{+} \,\}$
AllOf	::=	$\texttt{allof}(id) : \{\, \mathbf{Match}^{+} \,\}$
Match	::=	$\boldsymbol{f}_{\boldsymbol{MatchID}}(\textit{attribute value},\ \mathbf{AttrCat})$
Condition	::=	$\texttt{condition}(id) : \{\, \texttt{true} \,\}$
	|	$\texttt{condition}(id) : \{\, f^{\mathsf{bool}}(a_1, \ldots, a_n) \,\}$
comb_{id}	::=	do | po | dup | pud | fa | ooa | ldo | lpo
Effect	::=	permit | deny
$f_{MatchID}$	::=	string − equal | date − equal | anyURI − equal | *etc*
AttrCat	::=	subject | action | resource | environment| *etc*
	XACML Request Components	
Request	::=	$\texttt{request:}\ \{\, \mathbf{Attribute}^{+} \,\}$
Attribute	::=	$\mathbf{AttrCat}(\ \textit{attribute value}\ )$ | *external state*
	XACML Response Components	
Result	::=	$\texttt{decision:}\ \{\, \mathbf{Decision} \,\}$
Decision	::=	Permit | Deny | Indeterminate | NotApplicable

Listing 4.1: Syntax of XACML 3.0: PolicySet Element

```
<xs:element name="PolicySet" type="xacml:PolicySetType"/>
<xs:complexType name="PolicySetType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>
    <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xacml:PolicySet"/>
      <xs:element ref="xacml:Policy"/>
      <xs:element ref="xacml:PolicySetIdReference"/>
      <xs:element ref="xacml:PolicyIdReference"/>
      <xs:element ref="xacml:CombinerParameters"/>
      <xs:element ref="xacml:PolicyCombinerParameters"/>
      <xs:element ref="xacml:PolicySetCombinerParameters"/>
    </xs:choice>
    <xs:element ref="xacml:ObligationExpressions" minOccurs="0"
        />
    <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="PolicySetId" type="xs:anyURI" use="
      required"/>
  <xs:attribute name="Version" type="xacml:VersionType" use="
      required"/>
  <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use
      ="required"/>
  <xs:attribute name="MaxDelegationDepth" type="xs:integer" use=
      "optional"/>
</xs:complexType>
```

A PolicySet contains another PolicySet or Policy elements. XACML 3.0 provides two ways to define the included PolicySet (or Policy) elements. The enclosed PolicySet can be defined directly as a part of the body of the top PolicySet. Another way is to defined it indirectly by only including a reference to the other PolicySet element using its identifier. In order to avoid a bulky definition, we only accommodate the indirect way in this thesis. The same case applies to the enclosed Policy elements. We only accommodate indirect definition by using the Policy identifier as a reference to the enclosed Policy element.

Based on XACML 3.0 specification (see [Ris13] page 45), the `CombinerParameter`, `PolicyCombinerParameters`, and `PolicySetCombinerParameters` are optional entities. Thus, we omit this entities since they are not the core ones.

In this abstraction, we model a set of policies as a sequence. A PolicySet can have empty sequence of policies or a sequence of other PolicySets or Policies. The sequence of PolicySets (or Policies) is combined using a specific combining algorithm.

The abstract syntax of PolicySet is defined as follows.

$$\texttt{policyset}(\mathcal{PS}_{id}) : \{\, \text{comb}_{id} \;;\; \mathcal{T} \;;\; \langle p_1, p_2, \ldots, p_n \rangle \,\} \quad n \geq 0$$

A PolicySet contains the following attributes and elements:

- a PolicySet identifier $\mathcal{PS}_{id}$,
- a policy combining algorithm identifier comb_{id},
- a Target element $\mathcal{T}$, and
- a set of Policy or other PolicySet elements which is identified by a sequence of policy's identifiers $\langle p_1, p_2, \ldots, p_n \rangle$, $n \geq 0$.

4.1.2 Policy Element

The original syntax of Policy element defined in [Ris13] is as follows.

Listing 4.2: Syntax of XACML 3.0: Policy Element

```
<xs:element name="Policy" type="xacml:PolicyType"/>
  <xs:complexType name="PolicyType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>
    <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
      <xs:element ref="xacml:RuleCombinerParameters" minOccurs="
          0"/>
      <xs:element ref="xacml:VariableDefinition"/>
      <xs:element ref="xacml:Rule"/>
    </xs:choice>
    <xs:element ref="xacml:ObligationExpressions" minOccurs="0"
        />
    <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="PolicyId" type="xs:anyURI" use="required"
      />
  <xs:attribute name="Version" type="xacml:VersionType" use="
      required"/>
  <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="
      required"/>
  <xs:attribute name="MaxDelegationDepth" type="xs:integer" use=
      "optional"/>
</xs:complexType>
```

A Policy is the second layer of policy model in XACML. Similar to PolicySet, we model a set of Rules as a sequence. It is not allowed to have an empty sequence of Rules (based on XACML specification [Ris13] page 49). The sequence of Rules is combined with a specific combining algorithm.

The abstract syntax of Policy is defined as follows.

$$\texttt{policy}(\mathcal{P}_{id}) : \{\, \text{comb}_{id} \;;\; \mathcal{T} \;;\; \langle r_1, r_2, \ldots, r_n \rangle \,\} \quad n \geq 1$$

A Policy element contains

- a Policy identifier $\mathcal{P}_{id}$,
- a rule combining algorithm identifier comb_{id},
- a Target $\mathcal{T}$, and
- a set of Rule elements which is identified by a sequence of Rule's identifiers $\langle r_1, r_2, \ldots, r_n \rangle$, $n \geq 1$.

4.1.3 Rule Element

A Rule is the smallest policy entity in XACML and defines an individual rule in the policy. A Rule has only one effect: to Deny or to Permit an access. When the Rule's Target matches the Request, the applicability of the Rule is refined by a set of propositional formulae as written in the Rule's Condition.

The original syntax of Rule element defined in [Ris13] is as follows.

Listing 4.3: Syntax of XACML 3.0: Rule Element

```
<xs:element name="Rule" type="xacml:RuleType"/>
<xs:complexType name="RuleType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:Target" minOccurs="0"/>
    <xs:element ref="xacml:Condition" minOccurs="0"/>
    <xs:element ref="xacml:ObligationExpressions" minOccurs="0"
        />
    <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="RuleId" type="xs:string" use="required"/>
  <xs:attribute name="Effect" type="xacml:EffectType" use="
      required"/>
</xs:complexType>
```

The abstract syntax of Rule is defined as follows.

$$\texttt{rule}(\mathcal{R}_{id}) : \{\, E \;;\; \mathcal{T} \;;\; \mathcal{C} \,\}$$

A Rule element contains

- a Rule identifier $\mathcal{R}_{id}$,
- an effect $E \in \{\, \texttt{permit}, \texttt{deny} \,\}$,
- a Target $\mathcal{T}$, and
- a Condition $\mathcal{C}$.

4.1.4 Target Element

A Target identifies the set of decision requests that the parent element (that is, either a PolicySet, a Policy or a Rule) is intended to evaluate.

The original syntax of Target element defined in [Ris13] is as follows.

Listing 4.4: Syntax of XACML 3.0: Target Element

```
<xs:element name="Target" type="xacml:TargetType"/>
<xs:complexType name="TargetType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
  <xs:element ref="xacml:AnyOf"/>
  </xs:sequence>
</xs:complexType>
```

XACML 3.0 allows Target element to not having AnyOf element. We call this as an empty Target and it is denoted by null. An empty Target (indicated by null) always matches any requests. The Target contains a conjunctive sequence of AnyOf elements.

The abstract syntax of Target is defined as follows.

$$\texttt{target}(\mathcal{T}_{id}) : \{\, \mathsf{null} \,\}$$

or

$$\texttt{target}(\mathcal{T}_{id}) : \{\, \mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n \,\} \quad n \geq 1$$

A Target element contains

- a Target identifier $\mathcal{T}_{id}$,

– an empty body indicated by null, or

– a conjunctive sequence of AnyOf elements $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$, $n \geq 1$.

4.1.5 AnyOf Element

The original syntax of AnyOf element defined in [Ris13] is as follows.

Listing 4.5: Syntax of XACML 3.0: AnyOf Element

```
<xs:element name="AnyOf" type="xacml:AnyOfType"/>
<xs:complexType name="AnyOfType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element ref="xacml:AllOf"/>
  </xs:sequence>
</xs:complexType>
```

An AnyOf element contains a disjunctive sequence of AllOf elements The abstract syntax of AnyOf is defined as follows.

$$\texttt{anyof}(\mathcal{E}_{id}) : \{\, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \,\} \quad n \geq 1$$

An AnyOf element contains

– an AnyOf identifier $\mathcal{E}_{id}$ and

– a disjunctive sequence of AllOf elements $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, $n \geq 1$.

4.1.6 AllOf Element

The original syntax of AllOf element defined in [Ris13] is as follows.

Listing 4.6: Syntax of XACML 3.0: AllOf Element

```
<xs:element name="AllOf" type="xacml:AllOfType"/>
<xs:complexType name="AllOfType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element ref="xacml:Match"/>
  </xs:sequence>
</xs:complexType>
```

Each AllOf element contains a conjunctive sequence of Match elements. The abstract syntax of AllOf is defined as follows.

$$\texttt{allof}(\mathcal{A}_{id}) : \{\, \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n \,\} \quad n \geq 1$$

An AllOf element contains

- an AllOf identifier $\mathcal{A}_{id}$ and
- a conjunctive sequence of Match elements $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$, $n \geq 1$.

4.1.7 Match Element

A Match is the smallest element in Target. The original syntax of Match element defined in [Ris13] is as follows.

Listing 4.7: Syntax of XACML 3.0: Match Element

```
<xs:element name="Match" type="xacml:MatchType"/>
<xs:complexType name="MatchType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeValue"/>
    <xs:choice>
      <xs:element ref="xacml:AttributeDesignator"/>
      <xs:element ref="xacml:AttributeSelector"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

`AttributeDesignator` and `AttributeSelector` elements are used to retrieve a bag of attribute values in the request context, which comes from Context Handler. The Context Handler chooses values which has the same category defined in the AttributeDesignator or AttributeSelector. In this thesis, we only state the category which the Context Handler should find its corresponding values.

The abstract syntax of Match is defined as follows.

$$\boldsymbol{f_{MatchID}}(\mathit{attribute\ value},\ \mathit{AttrCat})$$

A Match contains a $f_{MatchID}$ attribute that specifies a function used in performing the match evaluation. The function $f_{MatchID}$ takes two arguments and

returns a result of type Boolean. The full list of $f_{MatchID}$ function can be seen in [Ris13] Appendix A. The first argument is an embedded value, provided by the Match element, and the second argument is an attribute value obtained from the Request element which has the same category as specified in attribute category *AttrCat*. There are four common attributes categories used in XACML, namely *subject* category, *action* category, *resource* category and *environment* category. The full list of attribute categories can be seen in [Ris13] Appendix B.

4.1.8 Condition Element

A Condition is a Boolean function over attributes or functions of attributes. The original syntax of Condition element defined in [Ris13] is as follows.

Listing 4.8: Syntax of XACML 3.0: Condition Element

```
<xs:element name="Condition" type="xacml:ConditionType"/>
 <xs:complexType name="ConditionType">
  <xs:sequence>
    <xs:element ref="xacml:Expression"/>
  </xs:sequence>
</xs:complexType>
```

An empty Condition is always associated to `true`. In this abstraction, the user is freely to define the Condition as long as its expression returns Boolean value, i.e., either true or false.

The abstract syntax of Condition is defined as follows.

$$\texttt{condition}(\mathcal{C}_{id}) : \{\, \texttt{true} \,\}$$

or

$$\texttt{condition}(\mathcal{C}_{id}) : \{\, f^{\mathsf{bool}}(a_1, \ldots, a_n) \,\} \quad n \geq 1$$

where each a_i is an attribute value.

A Condition element contains

- a Condition identifier $\mathcal{C}_{id}$,
- an empty body indicated by `true`, or
- a Boolean function $f^{\mathsf{bool}}(a_1, \ldots, a_n)$, $n \geq 1$, which its arguments are attribute values.

4.1.9 XACML Combining Algorithms.

Currently, XACML has twelve standard combining algorithms namely

- (unordered- and ordered-) deny-overrides (denoted by `do` in Table 4.1),
- (unordered- and ordered-) permit-overrides (`po`),
- deny-unless-permit (`dup`),
- permit-unless-deny (`pud`),
- first-applicable (`fa`),
- only-one-applicable (`ooa`),
- legacy (unordered- and ordered-) deny-overrides (`ldo`), and
- legacy (unordered- and ordered-) permit-overrides (`lpo`).

As explained before in Section 3.3, the behaviour of the ordered combining algorithms are identical to the unordered combining algorithms. The ordered combining algorithms might give different result with the unordered combining algorithms when the Obligations take place. Since we do not consider Obligations and Advices, we are safe to exclude ordered-deny-overrides combining algorithm, ordered-permit-overrides combining algorithm, legacy ordered-deny-overrides combining algorithm and legacy ordered-permit-overrides combining algorithm.

4.1.10 Example

Example 4.2 (Patient Record Policies) This example shows how to encode XACML components in our abstract syntax. The example is motivated from [Ris13] Section 4.3.2. The following plain-language rules are to be enforced:

Rule 1: A person, identified by his or her patient number, may read any record for which he or she is the designated patient.

Rule 2: A person may read any record for which he or she is the designated parent or guardian, and for which the patient is under 16 years of age.

Rule 3: A physician may write to any medical element for which he or she is the designated primary care physician, provided an email is sent to the patient.

Rule 4: An administrator shall not be permitted to read or write to medical elements of a patient record.

Listing 4.9: Patient Record Policies Code

```
policy(p1):{ do; target(p1):{ null }; < r1 > }
rule(r1):{ permit;
  target(r1): {
    anyof(r1any):{
      allof(r1all):{string_equal(read, action), anyURI_equal(urn
          :example:med:schemas:record, target_namespace),
          string_equal(patient, subject_role) } } };
   condition(r1):{ (patient_number(N) /\ patient_record(
    patient_id, X) /\ equal(N, X) )} }

policy(p2):{ do; target(p2):{ null }; < r2 > }
rule(r2):{ permit;
   target(r2):{
     anyof(r2any):{
       allof(r2all):{string_equal(read, action), anyURI_equal(
           urn:example:med:schemas:record, target_namespace),
           string_equal(parent_guardian, subject_role) } } };
   condition(r2):{ ( parent_guardian_id(P) /\ patient_record(
       parent_guardian_id, R) /\ equal(P, R) /\ patient_record(
       patient_dob, N) /\ current_date(T) /\ minus(T, N, Z) /\
       less_than(Z , 16 ) )} }

policy(p3):{ do; target(p3):{ null }; < r3 > }
rule(r3):{ permit;
  target(r3):{
    anyof(r3any):{
      allof(r3all):{string_equal(physician, subject_role),
          anyURI_equal(urn:example:med:schemas:record,
          target_namespace), string_equal(write, action) } } };
  condition(r3): { ( physician_id(P) /\ patient_record(
      patient_id, X) /\ patient_record(primaryCarePhysician, X,
      R) /\ equal( P, R ) )} }

policy(p4):{do; target(p4):{ null }; < r4 >}
rule(r4):{ deny;
  target(r4):{
    anyof(r4any1):{
      allof(r4all1):{ string_equal(administrator,subject_role) }
          },
    anyof(r4any2):{
      allof(r4all1):{ anyURI_equal(urn:example:med:schemas:
          record, target_namespace) } },
    anyof(r4any3):{
      allof(r4all31):{ string_equal(read, action) },
      allof(r4all32):{ string_equal(write, action) } } };
    condition(r4):{ true } }

policyset(ps1):{ do;
   target(ps1): {
      anyof(ps1any): {
```

```
37          allof(ps1all): { string_equal(urn:example:med:schemas:
                record, resource) } } };
38    < p1, p2, p3, p4 >}
```

In this example, each Rule describes a single rule as required in the policy specification and each Rule is encapsulated by a Policy (see lines $1-32$). Each Policy has an empty Target (indicated by null). Therefore, the relevant Target depends on the Target of each its inner Rule.

For the sake of clarity, we encode the *and* and *or* operators as binary operators denoted by $\wedge$ and $\vee$, respectively. Empty Condition, as shown in rule 4, line 32, is always denoted by true.

We omit the Obligation in Rule 3 since we do not consider Obligations in this thesis. At the end, there is a PolicySet, namely ps1 (line 34) which contains all of those four Policies described in the rules.

4.2 Abstract Syntax for XACML Request Components

A Request contains a set of Attributes giving further information about the access request that should agree with the policy's Target.

The original syntax of Request element defined in [Ris13] is as follows.

Listing 4.10: Syntax of XACML 3.0: Request Element

```
1 <xs:element name="Request" type="xacml:RequestType"/>
2 <xs:complexType name="RequestType">
3   <xs:sequence>
4     <xs:element ref="xacml:RequestDefaults" minOccurs="0"/>
5     <xs:element ref="xacml:Attributes" maxOccurs="unbounded"/>
6     <xs:element ref="xacml:MultiRequests" minOccurs="0"/>
7   </xs:sequence>
8   <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use=
        "required"/>
9 <xs:attribute name
```

The `ReturnPolicyIdList` entity is used to request the PDP to return a list of all applicable Policy PolicySet elements that were in the decision as a part of the decision response. This thesis focuses on modelling XACML policies (included its request element). Thus, entity is not relevant and therefore we

omit this entity. The `CombinedDecision` entity is used to request that the PDP combines multiple decisions into a single decision. We omit this entity since we only consider single request

Listing 4.11: Syntax of XACML 3.0: Attributes Element

```
<xs:element name="Attributes" type="xacml:AttributesType"/>
<xs:complexType name="AttributesType">
  <xs:sequence>
    <xs:element ref="xacml:Content" minOccurs="0"/>
    <xs:element ref="xacml:Attribute" minOccurs="0" maxOccurs="
        unbounded"/>
  </xs:sequence>
  <xs:attribute name="Category" type="xs:anyURI" use="required"
      />
  <xs:attribute ref="xml:id" use="optional"/>
</xs:complexType><xs:complexType name="SubjectType">
```

Listing 4.12: Syntax of XACML 3.0: Attribute Element

```
<xs:element name="Attribute" type="xacml:AttributeType"/>
<xs:complexType name="AttributeType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeValue" maxOccurs="unbounded"
    />
  </xs:sequence>
  <xs:attribute name="AttributeId" type="xs:anyURI" use="
      required"/>
  <xs:attribute name="Issuer" type="xs:string" use="optional"/>
  <xs:attribute name="IncludeInResult" type="xs:boolean" use="
      required"/>
</xs:complexType>
```

XACML 3.0 separates the Attribute category into sub-category using `AttributeId` entity. To simplify this model, we do not have this separations. Thus, the category element is a combination of XACML category and its subcategory. For example, in XACML 3.0, `subject` is a category and `subject-id` is `AttributeId`. It is enough to model them as a single category, i.e., `subject-id`.

The `IncludeInResult` entity is used to include the attribute in the result. The default value for this is false. This is useful to correlate requests with their responses in case of multiple requests. We omit this entity since we do not consider multi-request.

The set of attributes should not be empty since empty set means there is no request. An Attribute is the central abstraction of the request context. In this abstraction, there are two type of Attribute element. An Attribute can be an element with attribute category and its value or information about external

state, e.g., the current time, the temperature, etc. XACML 3.0 does not include external state into its Request element. The external state can be obtained as additional information that PEP provide to help the PDP makes decision, especially when the evaluation of Condition element needs extra information. Thus, for the simplicity, we add external state into our Request abstraction.

The abstract syntax of Request is defined as follows.

$$\texttt{request:}\ \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\} \quad n \geq 1$$

where each $\mathcal{A}_i$ is an Attribute defined as follows.

$$\mathit{AttrCat}(\mathit{attribute\ value}) \text{ or } \mathit{external\ state}$$

Example 4.3 (Example of XACML Request) This example is based on [Ris13] Section 4.2.2.

> Suppose that there is a request by the physician Julius Hibbert to read the patient date of birth in the record of Bartholomew Simpson.

When the context handler receives a request, first of all it retrieves all related information regarding the request. In this example, the Context Handler provides information regarding Julius' physical identifier and Bartholomew Simpson's record number. After all related information is gathered, the context handler sends a request in XACML format to PDP. The encoding of XACML 3.0 Request is as follows.

Listing 4.13: Example of XACML 3.0 Request Element

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Request
3     xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:
          wd-17
6     http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd
          -17.xsd"
7     ReturnPolicyIdList="false">
8
9    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-
         category:access-subject">
10     <Attribute IncludeInResult="false" AttributeId="urn:oasis:
           names:tc:xacml:1.0:subject:subject-id" Issuer="med.
           example.com">
11       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
             #string">
12         CN=Julius Hibbert
13       </AttributeValue>
```

```
14      </Attribute>
15      <Attribute IncludeInResult="false" AttributeId="urn:oasis:
            names:tc:xacml:3.0:example:attribute:role" Issuer="med.
            example.com">
16        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
              #string">
17          physician
18        </AttributeValue>
19      </Attribute>
20      <Attribute IncludeInResult="false" AttributeId="urn:oasis:
            names:tc:xacml:3.0:example:attribute:physician-id"
            Issuer="med.example.com">
21        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
              #string">
22          jh1234
23        </AttributeValue>
24      </Attribute>
25    </Attributes>
26
27    <Attributes
28      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
        resource">
29      <Content>
30        <md:record xmlns:md="urn:example:med:schemas:record"
31        xsi:schemaLocation="urn:example:med:schemas:record
32        http://www.med.example.com/schemas/record.xsd">
33        <md:patient>
34        <md:patientDoB>1992-03-21</md:patientDoB>
35        <md:patient-number>555555</md:patient-number>
36        <md:patientContact>
37          <md:email>b.simpson@example.com</md:email>
38        </md:patientContact>
39        </md:patient>
40        </md:record>
41      </Content>
42      <Attribute IncludeInResult="false" AttributeId="urn:oasis:
            names:tc:xacml:3.0:content-selector" >
43        <AttributeValue XPathCategory="urn:oasis:names:tc:xacml
              :3.0:attribute-category:resource" DataType=" urn:oasis
              :names:tc:xacml:3.0:data-type:xpathExpression">
44            md:record/md:patient/md:patientDoB
45        </AttributeValue>
46      </Attribute>
47      <Attribute IncludeInResult="false" AttributeId="urn:oasis:
            names:tc:xacml:2.0:resource:target-namespace" >
48        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
              #anyURI">
49          urn:example:med:schemas:record
50        </AttributeValue>
51      </Attribute>
52    </Attributes>
53
54    <Attributes
55      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
        action">
```

```
56      <Attribute IncludeInResult="false"
57        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id
              " >
58        <AttributeValue
59        DataType="http://www.w3.org/2001/XMLSchema#string">read</
        AttributeValue>
60        </Attribute>
61    </Attributes>
62
63    <Attributes
64      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:
        environment">
65      <Attribute IncludeInResult="false"
66        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:
              current-date" >
67        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
              #date"
68        >2010-01-11</AttributeValue>
69      </Attribute>
70    </Attributes>
71  </Request>
```

The encoding of Request element in our abstraction is as follows.

Listing 4.14: XACML Request Code

```
1  request:{
2     subject_id(julius_hilbbert),
3     subject_role(physician),
4     physician_id(jh1234),
5     patient_record(patient_dob, 1992-03-21),
6     patient_record(patient_number, 555555),
7     patient_record(patient_contact, b.simpson@example.com),
8     content_selector(patientDoB),
9     target_namespace(urn:example:med:schemas:record),
10    action(read),
11    current_date(2010-01-11)}
```

Lines 2–3 define the attribute category access-subject. Lines 5–7 define the attribute category resource. Lines 8–9 in specific define the content selector. Line 10 defines the action category and line 11 defines the environment category.

4.3 Abstract Syntax for XACML Response Component

A response element encapsulates the authorisation decision produced by the PDP.

The original syntax of Response element defined in [Ris13] is as follows.

Listing 4.15: Syntax of XACML 3.0: Response Element

```
<xs:element name="Response" type="xacml:ResponseType"/>
<xs:complexType name="ResponseType">
  <xs:sequence>
    <xs:element ref="xacml:Result" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Listing 4.16: Syntax of XACML 3.0: Result Element

```
<xs:complexType name="ResultType">
  <xs:sequence>
    <xs:element ref="xacml:Decision"/>
    <xs:element ref="xacml:Status" minOccurs="0"/>
    <xs:element ref="xacml:Obligations" minOccurs="0"/>
    <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
    <xs:element ref="xacml:Attributes" minOccurs="0" maxOccurs="
        unbounded"/>
    <xs:element ref="xacml:PolicyIdentifierList" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Listing 4.17: Syntax of XACML 3.0: Decision Element

```
<xs:element name="Decision" type="xacml:DecisionType"/>
<xs:simpleType name="DecisionType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Permit"/>
    <xs:enumeration value="Deny"/>
    <xs:enumeration value="Indeterminate"/>
    <xs:enumeration value="NotApplicable"/>
  </xs:restriction>
</xs:simpleType>
```

The abstract syntax for Response is defined as follows.

$$\texttt{decision:}\ \{\mathcal{D}\}$$

The decision $\mathcal{D}$ can be either Permit, Deny, Indeterminate, or NotApplicable.

Example 4.4 (Example of XACML Response) Previously, the PDP got a request from physician Julius Hibbert to read Bartholomew Simpson's date of birth from his patient record. Since there is no policy mentions about any right to read patient record for a physician, hence, the PDP returned NotApplicable.

Listing 4.18: Example of XACML 3.0 Response Element

```
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:
      wd-17
  http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd
      -17.xsd">
  <Result>
    <Decision>NotApplicable</Decision>
  </Result>
</Response>
```

Listing 4.19: XACML Response Code

```
decision:{ not-applicable }
```

Chapter 5

XACML 3.0 Semantics

> Do you wish me a good morning, or mean that it is a good morning whether I want it or not; or that you feel good this morning; or that it is a morning to be good on?
>
> J.R.R. Tolkien, *The Hobbit*

In this chapter, we describe the semantics of XACML 3.0 elements. First, we introduce component values of XACML in Section 5.1. Next, we explain the semantics for each XACML element in Section 5.2 starting from the Match evaluation continued to PolicySet evaluation and finally, we present the evaluation of XACML combining algorithms in Section 5.3. We present related work in Section 5.4

XACML is an evolving standard that changes as need arises; this is clearly shown by the evolution from XACML 2.0 into XACML 3.0. Hence further evolution is conceivable in case new combining operators need to be dealt with. We present two new combining algorithms in Section 5.5.

V_3	Match, AllOf, AnyOf, Target	Condition	Rule, Policy, PolicySet
$\top$	`match`	`true`	`applicable`
$\bot$	`notmatch`	`false`	`notapplicable`
I	`indeterminate`	`indeterminate`	`indeterminate`

Table 5.1: Mapping V_3 into XACML components.

5.1 Component Values of XACML

The result of evaluating the entities Match, AllOf, AnyOf and Target is one of `match`, `notmatch` or `indeterminate`. The `indeterminate` value indicates that the decision of whether or not a policy is applicable cannot be determined due to an error during evaluation.

The Rule evaluation depends on the Target evaluation and the Condition evaluation. The Condition element is a Boolean function over a set of attribute values or functions of attributes. This function returns either `true`, `false` or `indeterminate`. An empty Condition is always evaluated to `true`. The result of evaluating a Rule is either `applicable`, `notapplicable` or `indeterminate`. An applicable Rule has an effect that is either `deny` or `permit`.

Finally, the evaluation of the entities Policy and PolicySet are based on a combining algorithm of which the result can be either `applicable` (with its effect being either `deny` or `permit`), `notapplicable` or `indeterminate`.

To give a formal semantics of these intentions we find it helpful to introduce partially ordered sets.

Definition 5.1 (Three-Valued lattice) We define $\mathcal{L}_3 = \langle V_3, \leq \rangle$ to be the *three-valued lattice* where V_3 is the set $\{ \bot, I, \top \}$ and the ordering is given by $\bot \leq I \leq \top$.

The set $\{ \bot, I, \top \}$ are mapped to the informal explanations above as summarised in Table 5.1.

Definition 5.2 Given a subset S of lattice $\mathcal{L}$ we denote *the greatest lower bound* (glb) and *the least upper bound* (lub) of S by $\bigsqcap S$ and $\bigsqcup S$, respectively. Recall that $\bigsqcap \emptyset = \top$ and $\bigsqcup \emptyset = \bot$.

So far our semantic treatment does not distinguish an applicable policy that denies an access from an applicable policy that permits an access. To rectify this shortcoming we shall write

- $\top_{\mathsf{d}}$ for an applicable policy denying access (noting that `d` denotes `deny`), and
- $\top_{\mathsf{p}}$ for an applicable process permitting access (noting that `p` denotes `permit`).

Similar considerations apply to indeterminate values where we now record the permissions that potentially could have resulted if there had been no errors.

Definition 5.3 (Extended Indeterminate Values) The possible extended indeterminate values are [Ris13]:

- Indeterminate Deny (I_{d}): an indeterminate value arising from a policy which could have evaluated to `deny` but not `permit`, e.g., a Rule which evaluates to indeterminate and its effect is `deny`.
- Indeterminate Permit (I_{p}): an indeterminate value arising from a policy which could have evaluated to `permit` but not `deny`, e.g., a Rule which evaluates to indeterminate and its effect is `permit`.
- Indeterminate Deny Permit (I_{dp}): an indeterminate value arising from a policy which could have effect either `deny` or `permit`.

In situations where we find that the set V_3 offers too little discerning power we shall therefore make use of the set $V_6 = \{ \top_{\mathsf{d}}, \top_{\mathsf{p}}, I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}}, \bot \}$. It is convenient to define a function $\sigma : V_3 \times \{ \mathsf{d}, \mathsf{p} \} \to V_6$ to map a value in V_3 and possible effects into a value in V_6 given a particular Rule's effect $e \in \{ \mathsf{d}, \mathsf{p} \}$ as follows[1]:

$$\sigma(X, e) = \begin{cases} X & \text{if } X = \bot \\ X_e & \text{otherwise} \end{cases}$$

As an example, $\sigma(\top, \mathsf{d}) = \top_{\mathsf{d}}$.

[1] For simplicity, we denote `d` for `deny` and `p` for `permit`.

5.2 Evaluation of XACML Components

We now give a syntax directed definition of the semantics of the XACML components, again leaving the challenging case of defining the combining algorithms to the next section. We use the $[\![\cdot]\!]$ notation to map XACML elements into their values in V_3 or V_6.

5.2.1 Evaluation of Match into V_3

We assume that the way the system evaluates the request is captured by the $\texttt{eval}_M/2$ function. The system uses the $\texttt{eval}_M$ function to evaluate the request context in order to get attribute values in Request element that match with attribute category in Match element. If an operational error occurs during the evaluation process, then the result of the entire Match semantics is `indeterminate`.

Let *cat* be an attribute category and let $\mathcal{Q} = \texttt{request:}\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a Request element. The evaluation of the request context in order to get attribute values in Request element that match with attribute category is as follows.

$$\texttt{eval}_M(cat, \mathcal{Q}) = \begin{cases} \texttt{error} & \text{if an error occurs during the evaluation process} \\ \{ v_1, \ldots, v_m \} & \text{if } cat(v_i) \in \{ \mathcal{A}_1, \ldots, \mathcal{A}_n \}, 1 \leq i \leq m \leq n \end{cases}$$

Let $\mathcal{M} = f_{MatchID}(v, C)$ be a Match where $f_{MatchID}$ is an identifier of Match evaluation function, v is the embedded attribute value and C is an attribute category. Let $\mathcal{Q}$ be a Request element. The evaluation of Match is as follows.

$$[\![f_{MatchID}(v, cat)]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \texttt{eval}_M(cat, \mathcal{Q}) \neq \texttt{error} \text{ and} \\ & \quad \exists v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \top \\ \bot & \text{if } \texttt{eval}_M(cat, \mathcal{Q}) = \{\ \} \text{ or} \\ & \quad (\texttt{eval}_M(cat, \mathcal{Q}) \neq \texttt{error} \text{ and} \\ & \quad\ \forall v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \bot) \\ I & \text{if } \texttt{eval}_M(cat, \mathcal{Q}) = \texttt{error} \text{ or} \\ & \quad ((\forall v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') \neq \top) \text{ and} \\ & \quad\ (\exists v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = I)) \end{cases} \tag{5.1}$$

Remark: In the previous equation, there are two appearances of $f_{MatchID}$. The first one indicates the syntax element of Match component and the latter indicates the evaluation of $f_{MatchID}$ function in order to determine the value of Match semantics.

Example 5.1 (Example of Match Evaluation) Let us continue from the previous example on the request ordered by the physician Julius Hibbert. Recall that the XACML Request $\mathcal{Q}$ sent by the context handler is as follows.

```
request:{
    subject-id(julius_hilbbert),
    subject-role(physician),
    physician_id(jh1234),
    patient_record(patient_dob, 1992-03-21),
    patient_record(patient_number, 555555),
    patient_record(patient_contact, b.simpson@example.com),
    content_selector(patientDoB),
    target_namespace(urn:example:med:schemas:record),
    action(read),
    current-date(2010-01-11)}
```

The result of $\texttt{eval}_M(\mathsf{action}, \mathcal{Q})$ is $\{\, \mathsf{read} \,\}$ because $\mathsf{action}(\mathsf{read})$ is in Request $\mathcal{Q}$. Consider that the evaluation of $\mathsf{string} - \mathsf{equal}(\mathsf{read}, \mathsf{read})$ returns true ($\top$), then the result of $[\![\mathsf{string} - \mathsf{equal}(\mathsf{read}, \mathsf{action})]\!](\mathcal{Q})$ is $\top$.

Suppose we have a different scenario. There is an error, e.g., a connection failure when PDP is evaluating the request context. Consequently, the $\texttt{eval}_M(\mathsf{action}, \mathcal{Q})$ returns `error` and the result of $[\![\mathsf{string} - \mathsf{equal}(\mathsf{read}, \mathsf{action})]\!](\mathcal{Q})$ is I.

5.2.2 Evaluation of AllOf, AnyOf and Target into V_3

Let $\mathcal{M}$ be a Match, let $\mathcal{A}$ be an AllOf, let $\mathcal{E}$ be an AnyOf, let $\mathcal{T}$ be a Target and let $\mathcal{Q}$ be a Request.

Suppose that $\texttt{allof}(\mathcal{A}_{id}) : \{\, \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n \,\}$, $n \geq 1$ be a AllOf elements and each $\mathcal{M}_i$ is a Match element. The evaluation of AllOf $\mathcal{A}_{id}$ over Request $\mathcal{Q}$ is as follows.

$$[\![\mathcal{A}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{M}_i]\!] = \top \\ \bot & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{M}_i]\!] = \bot \\ I & \text{otherwise} \end{cases} \tag{5.2}$$

Suppose that $\texttt{anyof}(\mathcal{E}_{id}) : \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$, $n \geq 1$ be a AnyOf elements and each $\mathcal{A}_i$ is an AllOf element. The evaluation of AnyOf $\mathcal{E}_{id}$ over Request $\mathcal{Q}$ is as follows.

$$[\![\mathcal{E}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{A}_i]\!] = \top \\ \bot & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{A}_i]\!] = \bot \\ I & \text{otherwise} \end{cases} \tag{5.3}$$

Suppose that $\texttt{allof}(\mathcal{T}_{id}) : \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$, $n \geq 1$ be a Target elements and each $\mathcal{E}_i$ is an AnyOf element. The evaluation of Target $\mathcal{T}_{id}$ over Request $\mathcal{Q}$ is as follows.

$$[\![\mathcal{T}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{E}_i]\!] = \top \\ \bot & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{E}_i]\!] = \bot \\ I & \text{otherwise} \end{cases} \tag{5.4}$$

An empty Target– indicated by null – is always evaluated to $\top$.

Example 5.2 (Example of Target Evaluation) Let us use the previous Patient Record Policies from Example 4.2 to show the evaluation of Target based on Request $\mathcal{Q}$ from previous example. In this example, we only show the evaluation of Target from Rule `r4`. Let us recall this Rule as follows.

```
1  rule(r4):{ deny;
2    target(r4):{
3      anyof(r4any1):{
4        allof(r4all1):{ string_equal(administrator,subject-role) }
               },
5      anyof(r4any2):{
6        allof(r4all1):{ anyURI_equal(urn:example:med:schemas:
             record, target_namespace) }},
7      anyof(r4any3):{
8        allof(r4all31):{ string_equal(read, action },
9        allof(r4all32):{ string_equal(write, action }}};
10   condition(r4):{ true }}
```

We have shown that the evaluation of $[\![\mathsf{string\_equal(read, action)}]\!](\mathcal{Q})$ is equal to $\top$. Thus the evaluation of $[\![\mathsf{allof(r4all32}]\!](\mathcal{Q})$ is equal to $\top$ too. Hence the evaluation of $[\![\mathsf{r4any3}]\!](\mathcal{Q})$ is equal to $\top$.

The evaluation of Match $\mathsf{anyURI\_equal(patient\_record, target\_namespace)}$. The result of $[\![\mathsf{anyURI\_equal(patient\_record, target\_namespace)}]\!](\mathcal{Q})$ is $\top$ because

target_namespace(patient_record) is in $\mathcal{Q}$ and we assume the evaluation of anyURI_equal(patient_record, patient_record is equal to $\top$. Hence, the evaluation of $[\![\mathsf{r4any2}]\!](\mathcal{Q})$ is $\top$.

The evaluation of Match $\mathsf{string-equal(administrator, subject-role)}$. We get that the evaluation of $\mathsf{string-equal(administrator, physician)}$ returns false ($\bot$). Thus, the result of $[\![\mathsf{string-equal(administrator, subject-role)}]\!](\mathcal{Q})$ is $\bot$. Hence, the evaluation of $[\![\mathsf{r4all1}]\!](\mathcal{Q})$ is $\bot$. Moreover, the evaluation of $[\![\mathsf{r4any1}]\!](\mathcal{Q})$ is $\bot$ too. Finally, the evaluation of $[\![\mathsf{r4}]\!](\mathcal{Q})$ is $\bot$.

5.2.3 Evaluation of Condition into V_3

We define the conditional evaluation function $\mathtt{eval}_C$ as an unspecified function used to evaluate Condition to a value in V_3, given a Request component $\mathcal{Q}$. The evaluation of Condition is then defined as follows:

$$[\![\mathcal{C}_{id}]\!](\mathcal{Q}) = \mathtt{eval}_C(\mathcal{C}, \mathcal{Q}) \tag{5.5}$$

Example 5.3 (Example of Condition Evaluation) Suppose we have a Request from Bartholomew Simpson to read his own patient record. Following is the Request component.

```
request:{
    subject(bartholomew_simpson),
    subject-role(patient),
    patient_id(5555),
    resource_target(patient_record),
    patient_record(patient_id, 5555),
    action(read) }
```

By evaluating the Target element of each Rule in the previous example, we find out that only the Target of Rule r1 matches with the Request component. Recall that the Condition of r1 is as follows.

```
condition(r1):{
  patient_number(N) /\ patient_record(patient_id, X) /\ N = X }
```

By substituting the variables N and X with 5555 that we get from the Request component, the $\mathtt{eval}_C(\mathcal{C}, \mathcal{Q})$ returns $\top$.

5.2.4 Evaluation of Rule into V_6

Let $\texttt{rule}(\mathcal{R}_{id}) : \{\, E \;;\; \mathcal{T} \;;\; \mathcal{C} \,\}$ be a Rule where E is the Rule's effect, $E \in \{\, \texttt{permit}, \texttt{deny} \,\}$, $\mathcal{T}$ is a Target and $\mathcal{C}$ is a Condition. Let $\mathcal{Q}$ be a Request. Then, the evaluation of Rule $\mathcal{R}_{id}$ is determined as follows:

$$[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \begin{cases} \top_{\texttt{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \text{ and } E = \texttt{deny} \\ \top_{\texttt{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \text{ and } E = \texttt{permit} \\ I_{\texttt{d}} & \text{if } \big(([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = I) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = I\big) \text{ and} \\ & \quad E = \texttt{permit} \\ I_{\texttt{p}} & \text{if } \big(([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = I) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = I\big) \text{ and} \\ & \quad E = \texttt{deny} \\ \bot & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \bot\big) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \end{cases} \tag{5.6}$$

To increase our confidence in the semantics defined above it may be useful to define the evaluation of Rule in a different way using a new operator $\rightsquigarrow$: $V_3 \times V_3 \to V_3$. Let F and G be two values in V_3 and define:

$$F \rightsquigarrow G = \begin{cases} G & \text{if } F = \top \\ F & \text{otherwise} \end{cases}$$

Proposition 5.4 *Let* $\texttt{rule}(\mathcal{R}) : \{\, E \;;\; \mathcal{T} \;;\; \mathcal{C} \,\}$ *be a Rule and let* $\mathcal{Q}$ *be a Request. Let* e *be* $\texttt{p}$ *when the effect* E *is* $\texttt{permit}$ *and* e *be* $\texttt{d}$ *when the effect* E *is* $\texttt{deny}$*. Then, the following equation holds*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \sigma\,(\;\; [\![\mathcal{T}]\!](\mathcal{Q}) \rightsquigarrow [\![\mathcal{C}]\!](\mathcal{Q}), \quad e \;\;)$$

Proof. The table below tabulates all nine possibilities for $[\![\mathcal{R}]\!](\mathcal{Q})$, $[\![\mathcal{T}]\!](\mathcal{Q})$ and $[\![\mathcal{C}]\!](\mathcal{Q})$ and calculates these combinations. The equivalence of the last two columns constitutes the proof of Proposition 5.4.

$[\![\mathcal{T}]\!](\mathcal{Q})$	$[\![\mathcal{C}]\!](\mathcal{Q})$	$[\![\mathcal{T}]\!](\mathcal{Q}) \rightsquigarrow [\![\mathcal{C}]\!](\mathcal{Q})$	$\sigma([\![\mathcal{T}]\!](\mathcal{Q}) \rightsquigarrow [\![\mathcal{C}]\!](\mathcal{Q}), e)$	$[\![\mathcal{R}]\!](\mathcal{Q})$
$\top$	$\top$	$\top$	$\top_e$	$\top_e$
$\top$	$\bot$	$\bot$	$\bot$	$\bot$
$\top$	I	I	I_e	I_e
$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\bot$	I	$\bot$	$\bot$	$\bot$
I	$\top$	I	I_e	I_e
I	$\bot$	I	I_e	I_e
I	I	I	I_e	I_e

5.2.5 Evaluation of Policy into V_6

The standard evaluation of a Policy element is described in [Ris13] as follows.

Target	**Combining Algorithm Value**	**Evaluation of Policy**
"Match"	Any value	Specified by the combining algorithm
"No-match"	Don't care	"Not Applicable"
"Indeterminate"	"Not Applicable"	"Not Applicable"
	"Permit"	"Indeterminate Permit"
	"Deny"	"Indeterminate Deny"
	"Indeterminate"	"Indeterminate Deny Permit"
	"Indeterminate Deny Permit"	"Indeterminate Deny Permit"
	"Indeterminate Permit"	"Indeterminate Permit"
	"Indeterminate Deny"	"Indeterminate Deny"

To formalise this, let $\texttt{policy}(\mathcal{P}_{id}) : \{\, \text{comb}_{id} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ be a Policy where $\mathcal{T}$ is a Target and $\langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ is a sequence of Rules, and let $\mathcal{Q}$ be a Request. Then, the evaluation of Policy $\mathcal{P}_{id}$ is defined as follows (using abbre-

viations explained below):

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \begin{cases} \top_{\mathtt{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \top_{\mathtt{d}} \\ \top_{\mathtt{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \top_{\mathtt{p}} \\ I_{\mathtt{dp}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = I_{\mathtt{dp}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \big(I \text{ or } I_{\mathtt{dp}}\big)\big) \\ I_{\mathtt{d}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = I_{\mathtt{d}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \big(\top_{\mathtt{d}} \text{ or } I_{\mathtt{d}}\big)\big) \\ I_{\mathtt{p}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = I_{\mathtt{p}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \big(\top_{\mathtt{p}} \text{ or } I_{\mathtt{p}}\big)\big) \\ \bot & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \bot\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \bot\big) \end{cases} \tag{5.7}$$

where $\mathbb{R} = \langle [\![\mathcal{R}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{R}_n]\!](\mathcal{Q}) \rangle$.

Remark: The second equation: $I_{\mathtt{dp}}$ if $[\![\mathcal{T}]\!](\mathcal{Q}) = I$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = I$ will never happen. It is revised by the definition of combining algorithms. All combining algorithms never return I.

5.2.6 Evaluation of PolicySet into V_6

The evaluation of a PolicySet is similar to that of a Policy evaluation. However, the input of the combining algorithm is a sequence of either PolicySets or Policies. For the formal definition, let $\mathtt{policyset}(\mathcal{PS}_{id}) : \{\, \mathrm{comb}_{id} \;;\; \mathcal{T} \;;\; \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle \,\}$ be a PolicySet where $\mathcal{T}$ is a Target and $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ is a sequence of PolicySets (or Policies), and let $\mathcal{Q}$ be a Request. Then, the evaluation of PolicySet $\mathcal{PS}_{id}$ is

defined as follows:

$$[\![\mathcal{PS}_{id}]\!](\mathcal{Q}) = \begin{cases} \top_{\mathtt{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = \top_{\mathtt{d}} \\ \top_{\mathtt{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = \top_{\mathtt{p}} \\ I_{\mathtt{dp}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = I_{\mathtt{dp}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = \big(I \text{ or } I_{\mathtt{dp}}\big)\big) \\ I_{\mathtt{d}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = I_{\mathtt{d}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \big(\top_{\mathtt{d}} \text{ or } I_{\mathtt{d}}\big)\big) \\ I_{\mathtt{p}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = I_{\mathtt{p}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \big(\top_{\mathtt{p}} \text{ or } I_{\mathtt{p}}\big)\big) \\ \bot & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \bot\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \bot\big) \end{cases} \tag{5.8}$$

where $\mathbb{P} = \langle [\![\mathcal{P}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{P}_n]\!](\mathcal{Q})\rangle$.

5.3 Semantics of XACML Combining Algorithms

As mentioned before, there are twelve combining algorithms in XACML 3.0, namely (i) (unordered- and ordered-) deny-overrides (`do`), (ii) (unordered- and ordered-) permit-overrides (`po`), (iii) deny-unless-permit (`dup`), (iv) permit-unless-deny (`pud`), (v) first-applicable (`fa`), (vi) only-one-applicable (`ooa`), (vii) legacy (unordered- and ordered-) deny-overrides (`ldo`) and (viii) legacy (unordered- and ordered-) permit-overrides (`lpo`). In this thesis we do not consider ordered-deny-overrides combining algorithm, ordered-permit-overrides combining algorithm, legacy ordered-deny-overrides combining algorithm and legacy ordered-permit-overrides combining algorithm since we do not take into account Obligations and Advices. Hence, we only cover eight combining algorithms in this section.

The input for each combining algorithm is a sequence of PolicySet, Policy, or Rule values. There are some combining algorithms that keep track the set of extended indeterminate values while others do not. The combining algorithms which use the extended indeterminate values are: deny-overrides and permit-overrides while the combining algorithms which do not use the extended indeterminate values are: deny-unless-permit, permit-unless-deny, first-applicable, only-one-applicable, legacy deny-overrides, and legacy permit-overrides. The output of those combining algorithms which do not track the extended set of indeterminate values must be treated as $I_{\mathtt{dp}}$ when those combining algorithms return indeterminate value. On the other hand, each value of extended inde-

terminate value shall be mapped to I_{dp} when they come as an input of those combining algorithms.

To guard against modelling artefacts we also provide an alternative way of characterising the combining algorithms and we formally prove the equivalence of these approaches.

5.3.1 Pairwise Policy Values

Our main semantic domain for expressing the combining algorithms is built around V_6. It is to be equipped with a partial order, in much the same way that V_3 became $\mathcal{L}_3 = \langle V_3, \leq \rangle$, and the intended partial order is displayed in Figure 5.1. However, explanations on operations on the partially ordered set become more amenable by giving an isomorphic representation of the elements of V_6.

The isomorphic representation makes use of pairs where component values are chosen as follows. The value **1** represents an applicable value (either deny or permit), $\mathbf{\frac{1}{2}}$ represents an indeterminate value and **0** means that there is no applicable value. In each pair, the first component represents the deny value amount of `deny` and the second component represents permit value amount of `permit`. We define $[0,0]$ for not applicable ($\bot$) because neither deny nor permit is applicable, $[1,0]$ for applicable with deny effect ($\top_{\mathsf{d}}$) because only deny value is applicable, $[\frac{1}{2},0]$ for I_{d} because the deny part is indeterminate, $[\frac{1}{2},\frac{1}{2}]$ for I_{dp} because both deny and permit have indeterminate values. A similar encoding works for permit. This is summarised in Figure 5.1.

Definition 5.5 (Pairwise Policy Values) Formally, the set of *pairwise policy values* is $\mathbf{P} = \{\, [1,0], [0,1], [\frac{1}{2},0], [0,\frac{1}{2}], [\frac{1}{2},\frac{1}{2}], [0,0] \,\}$

We write $[D,P]$ for a typical element on $\mathbf{P}$ and denote $\mathsf{d}([D,P]) = D$ and $\mathsf{p}([D,P]) = P$ for the function that returns the deny value (first component) and permit value (second component), respectively. The relationship between V_6 and $\mathbf{P}$ can then be summarised by defining a function $\delta : V_6 \to \mathbf{P}$ as a mapping function that maps V_6 into $\mathbf{P}$ as follows:

$$\delta(X) = \begin{cases} [1,0] & X = \top_{\mathsf{d}} \\ [0,1] & X = \top_{\mathsf{p}} \\ [\frac{1}{2},0] & X = I_{\mathsf{d}} \\ [0,\frac{1}{2}] & X = I_{\mathsf{p}} \\ [\frac{1}{2},\frac{1}{2}] & X = I_{\mathsf{dp}} \\ [0,0] & X = \bot \end{cases}$$

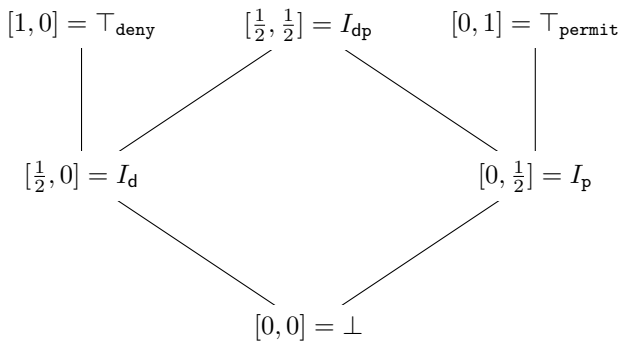

Figure 5.1: The partially ordered set $\boldsymbol{P}_{\mathbf{P}}$ (isomorphic to V_6) for pairwise policy values.

We set $\delta(\langle s_1, \ldots, s_n \rangle) = \langle \delta(s_1), \ldots, \delta(s_n) \rangle$ to extend δ to operate over sequences.

It is now straightforward to formally define the partial order $\sqsubseteq_{\mathbf{P}}$ on $\mathbf{P}$. We define $[D_1, P_1] \sqsubseteq_{\mathbf{P}} [D_2, P_2]$ if and only if $D_1 \leq D_2$ and $P_1 \leq P_2$ with $0 \leq \frac{1}{2} \leq 1$. We write $\boldsymbol{P}_{\mathbf{P}}$ for the partial ordered set (poset) $(\mathbf{P}, \sqsubseteq_{\mathbf{P}})$ illustrated in Figure 5.1.

The partial ordered set $\boldsymbol{P}_{\mathbf{P}}$ is not a lattice as many subsets have no least upper bound. However, to express those least upper bounds and greatest lower bounds that do exist we introduce the following functions. Let $max : 2^{\{1, \frac{1}{2}, 0\}} \to \{1, \frac{1}{2}, 0\}$ be the function that returns the maximum value of a non-empty subset of $\{1, \frac{1}{2}, 0\}$ and let $min : 2^{\{1, \frac{1}{2}, 0\}} \to \{1, \frac{1}{2}, 0\}$ be the function that returns the minimum value of a non-empty subset of $\{1, \frac{1}{2}, 0\}$.

This allows us to define $Max_{\sqsubseteq_{\mathbf{P}}} : 2^{\mathbf{P}} \to \mathbf{P}$ as a function that returns the maximum pairwise policy value defined as follows:

$$Max_{\sqsubseteq_{\mathbf{P}}}(S) = [max(\{0\} \cup \{d(X) \mid X \in S\}), max(\{0\} \cup \{p(X) \mid X \in S\})]$$

and $Min_{\sqsubseteq_{\mathbf{P}}} : 2^{\mathbf{P}} \to \mathbf{P}$ as a function that return the minimum pairwise policy value which is defined as follows:

$$Min_{\sqsubseteq_{\mathbf{P}}}(S) = [min(\{1\} \cup \{d(X) \mid X \in S\}), min(\{1\} \cup \{p(X) \mid X \in S\})]$$

One may observe that $Max_{\sqsubseteq_{\mathbf{P}}}(S)$ and $Min_{\sqsubseteq_{\mathbf{P}}}(S)$ as constructed above are not necessarily elements of $\boldsymbol{P}_{\mathbf{P}}$. However, if they are then $Max_{\sqsubseteq_{\mathbf{P}}}(S)$ denotes the least upper bound of S in the partially ordered set $\boldsymbol{P}_{\mathbf{P}}$, and $Min_{\sqsubseteq_{\mathbf{P}}}(S)$ denotes the greatest lower bound of S in the partially ordered set $\boldsymbol{P}_{\mathbf{P}}$. *Remark:* In

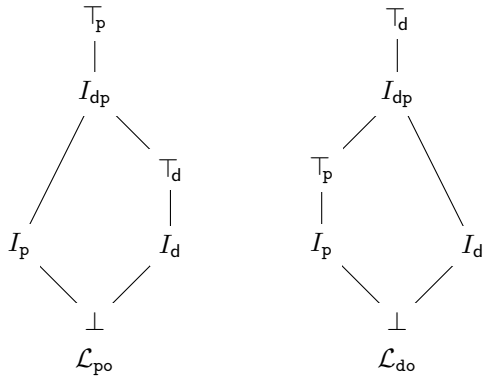

Figure 5.2: The lattice $\mathcal{L}_{po}$ for the Permit-Overrides Combining Algorithm (left); the lattice $\mathcal{L}_{do}$ for the Deny-Overrides Combining Algorithm (right).

fact, there are three values that may be constructed by these functions that fall outside of $\boldsymbol{P}_{\mathbf{P}}$: the values $[1, \frac{1}{2}]$, $[\frac{1}{2}, 1]$ and $[1, 1]$ denoting some form of conflict as we shall return to in Section 8.5.

5.3.2 The Permit-Overrides Combining Algorithm

We are now ready to deal with the first combining algorithm. The permit-overrides combining algorithm is intended for those cases where a permit decision should take priority over a deny decision. The behaviour of this algorithm is described in [Ris13] as follows:

1. If any decision is $\top_p$, then the result is $\top_p$,
2. otherwise, if any decision is I_{dp}, then the result is I_{dp},
3. otherwise, if any decision is I_p and another decision is I_d or $\top_d$, then the result is I_{dp},
4. otherwise, if any decision is I_p, then the result is I_p,
5. otherwise, if any decision is $\top_d$, then the result is $\top_d$,
6. otherwise, if any decision is I_d, then the result is I_d,
7. otherwise, the result is $\bot$.

As our first and most intuitive attempt at formalising this explanation we shall equip V_6 with a special partial order $\sqsubseteq_{\mathtt{po}}$ and define the permit-overrides combining algorithm as the least upper bound operator over the resulting lattice $\mathcal{L}_{\mathtt{po}} = (V_6, \sqsubseteq_{\mathtt{po}})$. The partial order $\sqsubseteq_{\mathtt{po}}$ is defined by the leftmost Hasse diagram in Figure 5.2 and it is immediate to verify that all pairs have a least upper bound and there is a least element and hence that $\mathcal{L}_{\mathtt{po}} = (V_6, \sqsubseteq_{\mathtt{po}})$ is a lattice.[2] The least upper bound operator for $\mathcal{L}_{\mathtt{po}}$ is denoted by $\bigsqcup_{\mathtt{po}}$.

To argue for this choice of $\mathcal{L}_{\mathtt{po}} = (V_6, \sqsubseteq_{\mathtt{po}})$ note that in the permit-overrides combining algorithm, the permit value takes the highest priority (see item 1 of the above description). Hence, the permit value (denoted by $\top_{\mathtt{p}}$) should be the top element. Whenever permit does not exists, we return $I_{\mathtt{dp}}$ if there is $I_{\mathtt{dp}}$ (see item 2). Hence, $I_{\mathtt{dp}}$ is bellow $\top_{\mathtt{p}}$. $I_{\mathtt{dp}}$ is also the join of $I_{\mathtt{p}}$ and $\top_{\mathtt{d}}$ or $I_{\mathtt{p}}$ and $I_{\mathtt{d}}$ (see item 3). However, $\top_{\mathtt{d}}$ is more favourable than $I_{\mathtt{d}}$ (see item 5 and 6), thus, $\top_{\mathtt{d}}$ should be on top of $I_{\mathtt{d}}$. The least element of this lattice is $\bot$ (see item 6). We are now ready to define the permit-overrides combining algorithm using the least upper bound operation $\bigsqcup_{\mathtt{po}}$.

Definition 5.6 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6 and let $\mathbf{S} = \{ s_1, \ldots, s_n \}$ be a set of policy values from $\mathbb{S}$. We define the *permit-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) = \bigsqcup_{\mathtt{po}} \mathbf{S} \tag{5.9}$$

Alternative Characterisation While we have argued that the previous construction is correct it is somewhat unsatisfactory in that the partial order $\sqsubseteq_{\mathtt{po}}$ is so heavily dependent on the policy combining algorithm. This motivates giving a definition directly in terms of the pairwise policy values $\mathbf{P}$. The idea is that we inspect the maximum value of `deny` and `permit` in the set of pairwise policy values. We conclude that the decision is permit if the `permit` is applicable (i.e. it has value 1). If the `permit` is indeterminate (i.e. it has value $\frac{1}{2}$) then the decision is $I_{\mathtt{dp}}$ when the `deny` is either indeterminate (i.e. it has value $\frac{1}{2}$) or applicable (i.e. it has value 1). Otherwise we take the maximum value of `deny` and `permit` from the set of pairwise policy values as the result of permit-overrides combining algorithm.

[2] It is well known that a partially ordered set is a complete lattice if and only if all least upper bounds exist, as the greatest lower bounds can then be constructed from the least upper bounds; in a similar manner, a finite partially ordered set is a lattice if and only if all binary least upper bounds exist and there is a least element, as the greatest lower bounds and greatest element can then be constructed from the binary least upper bounds and the least element.

Definition 5.7 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from $\mathbf{P}$ and let $\mathbf{S} = \{ s_1, \ldots, s_n \}$ be a set of pairwise policy values from $\mathbb{S}$. We define the *permit-overrides combining algorithm under pairwise policy values* $\mathbf{P}$ as follows:

$$\bigoplus_{\text{po}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [0,1] & \text{if } Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) = [D,1], D \geq 0 \\ [\frac{1}{2},\frac{1}{2}] & \text{if } Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) = [D,\frac{1}{2}], D \geq \frac{1}{2} \\ Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) & \text{otherwise} \end{cases} \tag{5.10}$$

One may check that $\bigoplus_{\text{po}}^{\mathbf{P}}$ only produces values that are in pairwise policy values $\mathbf{P}$.

We have now given two independent formalisations of the permit-overrides combining algorithm. The following result establishes their equivalence and thereby increases our faith in either one.

Proposition 5.8 *Let* $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ *be a sequence of policy values from* V_6*. Then*

$$\delta(\bigoplus_{\text{po}}^{V_6}(\mathbb{S})) = \bigoplus_{\text{po}}^{\mathbf{P}}(\delta(\mathbb{S}))$$

We refer to the Appendix A for the proof.

5.3.3 The Legacy Permit-Overrides Combining Algorithm

The legacy permit-overrides combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. The behaviour of the legacy permit-overrides combining algorithm is similar to the permit-overrides combining algorithm except that the legacy permit-overrides combining algorithm treats all extended indeterminate values as I_{dp}. The algorithm has the following behaviour.

1. If any rule evaluates to $\top_{\mathsf{p}}$, then the result is $\top_{\mathsf{p}}$.
2. Otherwise, if any rule having effect I_{p} or I_{dp}, then the result is I_{dp}.
3. Otherwise, if any rule evaluate to $\top_{\mathsf{d}}$, then the result is $\top_{\mathsf{d}}$.
4. Otherwise, if any rule evaluate to I_{d}, then the result is I_{dp}.
5. Otherwise, the result is $\bot$.

We simply model the legacy permit-overrides combining algorithm based on the permit-overrides combining algorithm since it behaves similarly to the permit-overrides combining algorithm. We add additional condition that all extended indeterminate values are mapped to $I_{\mathtt{dp}}$. The same case is applied for our alternative characteristic using pairwise policy values.

Definition 5.9 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *legacy permit-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{lpo}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathtt{dp}} & \text{if } \bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) \in \{\, I_{\mathtt{d}}, I_{\mathtt{p}}, I_{\mathtt{dp}} \,\} \\ \bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) & \text{otherwise} \end{cases} \tag{5.11}$$

Definition 5.10 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from $\mathbf{P}$. We define the *legacy permit-overrides combining algorithm under pairwise policy values* $\mathbf{P}$ as follows:

$$\bigoplus_{\mathtt{lpo}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [\frac{1}{2}, \frac{1}{2}] & \text{if } \bigoplus_{\mathtt{po}}^{\mathbf{P}}(\mathbb{S}) \in \{\, [\frac{1}{2}, 0], [0, \frac{1}{2}], [\frac{1}{2}, \frac{1}{2}] \,\} \\ \bigoplus_{\mathtt{po}}^{\mathbf{P}}(\mathbb{S}) & \text{otherwise} \end{cases} \tag{5.12}$$

In this case the two definitions are so similar that we hardly gain any extra assurances by providing both sets of characterisation; however, we decided to do so for completeness sake.

5.3.4 The Deny-Overrides Combining Algorithm

The deny-overrides combining algorithm is largely dual to that of the permit-overrides combining algorithm. It is intended for those cases where a deny decision should have priority over a permit decision. In [Ris13] the behaviour of this algorithm is described by:

1. If any decision is $\top_{\mathtt{d}}$ then the result is $\top_{\mathtt{d}}$,
2. otherwise, if any decision is $I_{\mathtt{dp}}$ then the result is $I_{\mathtt{dp}}$,
3. otherwise, if any decision is $I_{\mathtt{d}}$ and another decision is $I_{\mathtt{p}}$ or $\top_{\mathtt{p}}$, then the result is $I_{\mathtt{dp}}$,
4. otherwise, if any decision is $I_{\mathtt{d}}$, then the result is $I_{\mathtt{d}}$,
5. otherwise, if any decision is $\top_{\mathtt{p}}$, then the result is $\top_{\mathtt{p}}$,
6. otherwise, if any decision is $I_{\mathtt{p}}$, then the result is $I_{\mathtt{p}}$,

7. otherwise, the result is $\bot$.

Formally, we write $\mathcal{L}_{\mathsf{do}} = (V_6, \sqsubseteq_{\mathsf{do}})$ for the lattice with $\sqsubseteq_{\mathsf{do}}$ being the partial ordering depicted in the middle Hasse diagram of Figure 5.2. The least upper bound operator for $\mathcal{L}_{\mathsf{do}}$ is denoted by $\bigsqcup_{\mathsf{do}}$.

Definition 5.11 Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *deny-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathsf{do}}^{V_6}(\langle s_1, \ldots, s_n \rangle) = \bigsqcup_{\mathsf{do}} \{\, s_1, \ldots, s_n \,\} \tag{5.13}$$

Alternative Characterisation The deny-overrides combining algorithm can be expressed using pairwise policy values $\mathbf{P}$ in much the same way as was done for the permit-overrides combining algorithm by symmetry.

Definition 5.12 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from $\mathbf{P}$ let $\mathbf{S} = \{\, s_1, \ldots, s_n \,\}$ be a set of pairwise policy values from $\mathbb{S}$. We define the *deny-overrides combining algorithm under pairwise policy values* $\mathbf{P}$ as follows:

$$\bigoplus_{\mathsf{do}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [1,0] & \text{if } Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) = [1, P], P \geq 0 \\ [\frac{1}{2}, \frac{1}{2}] & \text{if } Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) = [\frac{1}{2}, P], P \geq \frac{1}{2} \\ Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) & \text{otherwise} \end{cases} \tag{5.14}$$

As before we can show that the two characterisations agree thereby increasing our faith in either one.

Proposition 5.13 *Let* $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ *be a sequence of policy values from* V_6. *Then*

$$\delta(\bigoplus_{\mathsf{do}}^{V_6}(\mathbb{S})) = \bigoplus_{\mathsf{do}}^{\mathbf{P}}(\delta(\mathbb{S}))$$

The proof of Proposition 5.13 is similar to the proof of Proposition 5.8 as should not be surprising given the symmetry between the permit-overrides combining algorithm and the deny-overrides combining algorithm.

5.3.5 The Legacy Deny-Overrides Combining Algorithm

As likely the deny-overrides combining algorithm, the legacy deny-overrides combining algorithm is a dual of the legacy permit-overrides combining algorithm. The behaviour of this algorithm is as follows.

1. If any rule evaluates to $\top_{\mathsf{d}}$, then the result is $\top_{\mathsf{d}}$.
2. Otherwise, if any rule having effect I_{d} or I_{dp}, then the result is I_{dp}.
3. Otherwise, if any rule evaluate to $\top_{\mathsf{p}}$, then the result is $\top_{\mathsf{p}}$.
4. Otherwise, if any rule evaluate to I_{p} or I_{dp}, then the result is I_{dp}.
5. Otherwise, the result is $\bot$.

Definition 5.14 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *legacy deny-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{ldo}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathsf{dp}} & \text{if } \bigoplus_{\mathtt{do}}^{V_6}(\mathbb{S}) \in \{\, I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}} \,\} \\ \bigoplus_{\mathtt{do}}^{V_6}(\mathbb{S}) & \text{otherwise} \end{cases} \tag{5.15}$$

Definition 5.15 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from $\mathbf{P}$. We define the *legacy deny-overrides combining algorithm under pairwise policy values* $\mathbf{P}$ as follows:

$$\bigoplus_{\mathtt{ldo}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [\frac{1}{2}, \frac{1}{2}] & \text{if } \bigoplus_{\mathtt{do}}^{\mathbf{P}}(\mathbb{S}) \in \{\, [\frac{1}{2}, 0], [0, \frac{1}{2}], [\frac{1}{2}, \frac{1}{2}] \,\} \\ \bigoplus_{\mathtt{do}}^{\mathbf{P}}(\mathbb{S}) & \text{otherwise} \end{cases} \tag{5.16}$$

5.3.6 The Deny-Unless-Permit Combining Algorithm

The deny-unless-permit combining algorithm is intended for those cases where a permit decision should have priority over a deny decision and an indeterminate or not-applicable values must never be the result. It is useful at the top level in a policy structure to ensure that a PDP will always return a definite answer, i.e., either permit or deny decision. The algorithm has the following behaviour:

1. If any decision is $\top_{\mathsf{p}}$, then the result is $\top_{\mathsf{p}}$.
2. Otherwise, the result is $\top_{\mathsf{d}}$.

Definition 5.16 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *deny-unless-permit combining algorithm under* V_6 as follows:

$$\bigoplus_{\text{dup}}^{V_6}(\mathbb{S}) \begin{cases} \top_{\mathsf{p}} & \text{if } \exists i \in \{1, \ldots, n\} : s_i = \top_{\mathsf{p}} \\ \top_{\mathsf{d}} & \text{otherwise} \end{cases} \tag{5.17}$$

Definition 5.17 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from *policyvalues*. We define the *deny-unless-permit combining algorithm under pairwise policy values* $\mathbf{P}$ as follows:

$$\bigoplus_{\text{dup}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [0,1] & \text{if } \exists i \in \{1, \ldots, n\} : s_i = [0,1] \\ [1,0] & \text{otherwise} \end{cases}$$

5.3.7 The Permit-Unless-Deny Combining Algorithm

The permit-unless-deny combining algorithm has the same behaviour as the deny-unless-permit combining algorithm. The different is only that that the deny decision has higher priority than the permit one. The behaviour of the permit-unless-deny combining algorithm is as follows.

1. If any decision is $\top_{\mathsf{d}}$, then the result is $\top_{\mathsf{d}}$.

2. Otherwise, the result is $\top_{\mathsf{p}}$.

Definition 5.18 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *deny-unless-permit combining algorithm under* V_6 as follows:

$$\bigoplus_{\text{pud}}^{V_6}(\mathbb{S}) = \begin{cases} \top_{\mathsf{d}} & \text{if } \exists i \in \{1, \ldots, n\} : s_i = \top_{\mathsf{d}} \\ \top_{\mathsf{p}} & \text{otherwise} \end{cases} \tag{5.18}$$

Definition 5.19 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from $\mathbf{P}$. We define the *deny-unless-permit combining algorithm under* V_6 as follows:

$$\bigoplus_{\text{pud}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [1,0] & \text{if } \exists i \in \{1, \ldots, n\} : s_i = [1,0] \\ [0,1] & \text{otherwise} \end{cases} \tag{5.19}$$

5.3.8 The First-Applicable Combining Algorithm

The result of the first-applicable algorithm is described as the first Rule, Policy or PolicySet element in the sequence whose Target and Condition is applicable. The pseudo-code of the first-applicable combining algorithm in XACML 3.0 [Ris13] makes it clear that the result of this algorithm is the first Rule, Policy or PolicySet that does *not* give the value "not applicable". This reflects the idea that an indeterminate policy could turn out to be an applicable policy and hence be the one to be chosen. The first-applicable combining algorithm does not use extended indeterminate values. Hence, all of extended indeterminate values (i.e., $I_{\mathsf{d}}, I_{\mathsf{p}}$ and I_{dp}) are treated equivalently as I_{dp}.

The first-applicable combining algorithm under policy values from V_6 and pairwise policy values from $\mathbf{P}$ are defined below.

Definition 5.20 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *first-applicable combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{fa}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathsf{dp}} & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}} \} \wedge \forall j : (j < i) \Rightarrow (s_j = \bot) \\ s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ \top_{\mathsf{p}}, \top_{\mathsf{d}} \} \wedge \forall j : (j < i) \Rightarrow (s_j = \bot) \\ \bot & \text{otherwise} \end{cases} \tag{5.20}$$

Definition 5.21 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from $\mathbf{P}$. We define the *first applicable combining algorithm under* $\mathbf{P}$ as follows:

$$\bigoplus_{\mathtt{fa}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [\frac{1}{2}, \frac{1}{2}] & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ [\frac{1}{2}, 0], [0, \frac{1}{2}], [\frac{1}{2}, \frac{1}{2}] \} \wedge \\ & \quad \forall j \in \{1,\ldots,n\} : (j < i) \Rightarrow (s_j = [0,0]) \\ s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ [1,0], [0,1] \} \wedge \\ & \quad \forall j \in \{1,\ldots,n\} : (j < i) \Rightarrow (s_j = [0,0]) \\ [0,0] & \text{otherwise} \end{cases} \tag{5.21}$$

5.3.9 The Only-One-Applicable Combining Algorithm

The only-one-applicable combining algorithm is described in [Ris13] in the following manner:

> *In the entire set of policies in the policy set, if no policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm shall be "Not Applicable". If more than one*

> *policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm shall be "Indeterminate".*
>
> *If only one policy is considered applicable by evaluation of its target, then the result of the policy-combining algorithm shall be the result of evaluating the policy. If an error occurs while evaluating the target of a policy, or a reference to a policy is considered invalid or the policy evaluation results in "Indeterminate", then the policy set shall evaluate to "Indeterminate", with the appropriate error status.*

Inspecting the pseudo-code provided in [Ris13] allows us to conclude that:

- If there are at least two applicable policies, then the result is indeterminate,
- otherwise, if there is the only one policy that is applicable, the result is its policy value,
- otherwise, the result is not applicable.

Please remember that the only-one-applicable combining algorithm does not track the set of extended indeterminate values. Hence, all of extended indeterminate values are mapped to I_{dp}.

Definition 5.22 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *only-one-applicable combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathsf{ooa}}^{V_6}(\mathbb{S}) = \begin{cases} s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{\top_{\mathsf{d}}, \top_{\mathsf{p}}\} \wedge \\ & \quad \forall j \in \{1,\ldots,n\} : j \neq i \Rightarrow s_j = \bot \\ I_{\mathsf{dp}} & \text{if } \big(\exists i \in \{1,\ldots,n\} : s_i \in \{I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}}\}\big) \vee \\ & \quad \big(\exists i,j \in \{1,\ldots,n\} : i \neq j \wedge s_i, s_j \in \{\top_{\mathsf{d}}, \top_{\mathsf{p}}\}\big) \\ \bot & \text{otherwise} \end{cases} \tag{5.22}$$

Alternative Characterisation In line with our previous developments we are also going to show how to express the only-one-applicable combining algorithm directly using pairwise policy values **P**. The idea is that we inspect the maximum value of `deny` and `permit` returned from the given set of pairwise policy values. By inspecting the maximum value for each element, we know exactly the combination of pairwise policy values i.e., if we find that both `deny` and `permit` are not 0, it means that the `deny` value and the `permit` value are

either applicable (i.e. it has value 1) or indeterminate (i.e. it has value $\frac{1}{2}$). Thus, the result of this algorithm is $I_{\mathtt{dp}}$ (based on the only-one-combining algorithm description above). However if only one element is not 0 then there is a possibility that many policies have the same applicable (or indeterminate) values. If there are at least two policies with the amount of `deny` (or `permit`) are either applicable or indeterminate value, then the result is $I_{\mathtt{dp}}$. Otherwise we take the maximum value of `deny` and `permit` from the given set of pairwise policy values as the result of only-one-applicable combining algorithm.

Definition 5.23 Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from $\mathbf{P}$ and let $\mathbf{S} = \{ s_1, \ldots, s_n \}$ be a set of pairwise policy values from $\mathbb{S}$. We define the *only-one-applicable combining algorithm under pairwise policy values* $\mathbf{P}$ as follows

$$\bigoplus_{\mathtt{ooa}}^{\mathbf{P}}(\mathbb{S}) = \begin{cases} [\frac{1}{2}, \frac{1}{2}] & \text{if } \big(Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) = [D, P], \\ & \quad (D = \frac{1}{2} \vee P = \frac{1}{2}) \vee (D = P = 1)\big) \vee \\ & \quad \big(\exists i, j \in \{ 1, \ldots, n \} : i \neq j, \\ & \quad\quad Min_{\sqsubseteq_{\mathbf{P}}}(\{ s_i, s_j \}) = [D, P], \\ & \quad\quad max(\{ D, P \}) \geq \frac{1}{2}\big) \\ Max_{\sqsubseteq_{\mathbf{P}}}(\mathbf{S}) & \text{otherwise} \end{cases} \tag{5.23}$$

We conclude our treatment of policy combining algorithms by showing that also in the case of the only-one-applicable combining algorithm do we achieve a full agreement between the two developments.

Proposition 5.24 *Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. Then*

$$\delta(\bigoplus_{\mathtt{ooa}}^{V_6}(\mathbb{S})) = \bigoplus_{\mathtt{ooa}}^{\mathbf{P}}(\delta(\mathbb{S}))$$

We refer to the Appendix A for the proof.

5.4 Related Work

A formalisation of XACML 2.0 using a multi-valued approach is done by Bruns et al. and Ni et al. using the Belnap 4-valued logic [BDH07, BH08] and $\mathcal{D}$-algebra 8-valued logic [NBL09], respectively. We consider analysing more detail Belnap 4-valued logic and $\mathcal{D}$-Algebra 8-valued logic [NBL09] as these are the formalisations closest in spirit to ours as well as being the ones closest to the

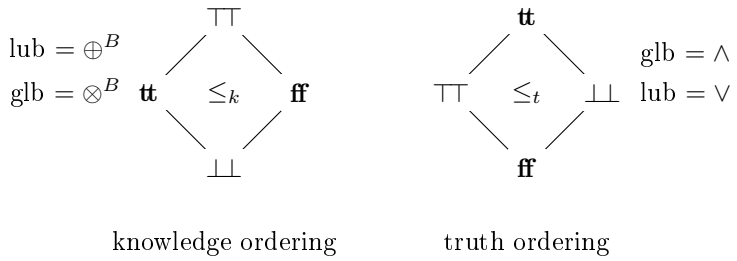

Figure 5.3: The bi-lattice of Belnap 4-Valued Logic.

current version of XACML. There are some mistakes in both cases and hence, they are not adequate to model XACML semantics.

5.4.1 XACML Semantics under Belnap 4-Valued Logic

Belnap [Bel77] defines a four-valued logic over the logical values **Four** = $\{ \top\top, \mathbf{tt}, \mathbf{ff}, \bot\bot \}$. The idea is that **tt** denotes true, **ff** denotes false, $\bot\bot$ denotes a missing truth value, and that $\top\top$ denotes a conflicting truth value. The four-valued logic is given the structure of a bi-lattice by equipping it with two partial orderings. One is the truth ordering ($\leq_t$) that may be viewed as extending the ordering given by 2-valued implication to operate over the new logical values. The other is the knowledge ordering ($\leq_k$) under which true and false are incomparable. We dispense with a formal definition and simply refer to the Hasse diagrams displayed in Figure 5.3. It is straightforward to check that **Four** is a lattice under both orderings. We use the familiar $\vee$ to denote least upper bound with respect to the truth ordering ($\leq_t$) and $\wedge$ to denote the greatest lower bound. For the knowledge ordering ($\leq_k$) we use $\oplus^B$ to denote least upper bound and $\otimes^B$ to denote greatest lower bound. To express the desired functions over **Four** one may define auxiliary operations and we mention three defined in [BH08]:

- An overwriting operator $[y \mapsto z]$ with $y, z \in$ **Four**; here $x[y \mapsto z]$ yields x if $x \neq y$, and z otherwise.
- A priority operator $x > y$; this is merely syntactic sugar for $x[\bot\bot \mapsto y]$.

Several papers have considered using Belnap 4-valued logic to formalise the meaning of operators on access control policies. A key study is that of Bruns

et al. considering the policy language PBel [BDH07, BH08] whereas that of Hankin et al. consider the design of an aspect oriented policy language AspectKB [HNN09]. In these approaches the value **tt** is interpreted as permitting the access, the value **ff** is interpreted as denying the access, $\bot\!\bot$ is interpreted as missing information, and $\top\!\top$ is interpreted as conflicting information. The approaches differ in their way of determining whether or not to permit access in case of the new logical values $\bot\!\bot$ and $\top\!\top$; this is connected to how the Policy Enforcement Point (PEP) interprets the new logical values but this is not discussed[3] in [BDH07, BH08, HNN09].

Bruns et al. attempt to formalise XACML 2.0 using their PBel approach. They consider three XACML policy combining operators and formalise them as follows [BH08]:

- **permit-overrides**(x, y) is modelled as $(x \oplus^B y)[\top\!\top \mapsto \mathbf{ff}]$.
- **first-applicable**(x, y) is modelled as $x > y$.
- **only-one-applicable**(x, y) is modelled as $(x \oplus^B y) \oplus^B ((x \oplus^B \neg y) \otimes^B (y \oplus^B \neg y))$[4].

Further, they suggested that the indeterminate value is treated as $\top\!\top$ whereas the inapplicable value is treated as $\bot\!\bot$.

However, modelling indeterminate as $\top\!\top$ has as a consequence that the permit-overrides combining operator is not defined correctly. Suppose we have two policies P_1 and P_2 where P_1 evaluates to permit (**tt**) and P_2 evaluates to indeterminate ($\top\!\top$). The result of the permit-overrides combining operator then is as follows:

$$\begin{aligned}(P_1 \oplus^B P_2)[\top\!\top \mapsto \mathbf{ff}] &= (\mathbf{tt} \oplus^B \top\!\top)[\top\!\top \mapsto \mathbf{ff}] \\ &= \top\!\top[\top\!\top \mapsto \mathbf{ff}] \\ &= \mathbf{ff}\end{aligned}$$

In other words, based on Bruns et al. definition using Belnap 4-valued logic, the result is deny (**ff**). This conflicts with the intentions of both XACML 2.0 and XACML 3.0 where the result of the permit-overrides combining operator should be permit (**tt**).

The problems stem from the fact that Bruns et al. define an indeterminate value as a conflict by formalising it as $\top\!\top$. However, they also say that sometimes indeterminate should be treated as $\bot\!\bot$ and sometimes as $\top\!\top$ [BH08], but they

[3] It would seem that Bruns et al. favour a so-called deny-based PEP whereas Hankin et al. favour a so-called permit-based PEP.

[4] $\neg p$ returns permit if and only if p is deny (and vice versa).

give no further explanation about the circumstances under which indeterminate is treated as $\top\!\!\top$ or as $\bot\!\!\bot$. Treating indeterminate as $\top\!\!\top$ is too strong because an indeterminate value does not always contain information about deny and permit at the same time. Only I_{dp} contains information about both deny and permit, whereas I_{d} and I_{p} only contain information only about either deny or permit. On the other hand, treating indeterminate as $\bot\!\!\bot$ is too weak because an indeterminate value is treated as not applicable despite the fact that there is information contained inside an indeterminate value.

We conclude that the Belnap 4-valued logic has no explicit definition of indeterminate. However, the Belnap 4-valued logic has a *conflict* value (i.e. $\top\!\!\top$) which is not a part of the XACML standard. Thus the match between XACML and Belnap 4-valued logic is rather imperfect.

5.4.2 XACML Semantics under $\mathcal{D}$-Algebra

Another approach to the formalisation of XACML was performed by Ni et al. in [NBL09] using a $\mathcal{D}$-algebra and operations over it. The carrier of the $\mathcal{D}$-algebra of interest consists of three kinds of decision values. The *deterministic decisions* are permit ($\{\mathbf{p}\}$), deny ($\{\mathbf{d}\}$) and not applicable ($\{\frac{\mathbf{n}}{\mathbf{a}}\}$), corresponding to our $\top_{\mathsf{p}}$, $\top_{\mathsf{d}}$ and $\bot$. The *non-deterministic decisions* are $\{\, \mathbf{d}, \frac{\mathbf{n}}{\mathbf{a}} \,\}$, $\{\, \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \,\}$, and $\{\, \mathbf{d}, \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \,\}$, corresponding to our I_{d}, I_{p} and I_{dp}. Finally, there are the special values $\emptyset$ and $\{\, \mathbf{p}, \mathbf{d} \,\}$; here $\emptyset$ is defined for an empty policy (or the absence of a policy) and $\{\, \mathbf{p}, \mathbf{d} \,\}$ for a conflict. We call the $\mathcal{D}$-algebra an 8-valued logic because the cardinality of the decisions D is $|\mathcal{P}(\{\, \mathbf{p}, \mathbf{d}, \frac{\mathbf{n}}{\mathbf{a}} \,\})| = 8$.

A $\mathcal{D}$-algebra is a carrier D (as explained above) equipped with an interpretation of the following operations [NBL09]:

- a constant 0 interpreted as $\emptyset$
- an unary operator $\neg$ interpreted as $\{\, \mathbf{p}, \mathbf{d}, \frac{\mathbf{n}}{\mathbf{a}} \,\} \setminus x$ (where $x \in D$)
- a binary operator $\oplus^{\mathcal{D}}$ interpreted as $x \cup y$ (where $x, y \in D$)
- a binary operator $\otimes^{\mathcal{D}}$ interpreted as $x \otimes^{\mathcal{D}} y = \begin{cases} \neg 0 & : x = y \\ 0 & : x \neq y \end{cases}$ (where $x, y \in D$)

To express the functions of interest one may define derived operations as for example $x \odot^{\mathcal{D}} y = \neg(\neg x \oplus^{\mathcal{D}} \neg y)$ and $x \ominus^{\mathcal{D}} y = x \odot^{\mathcal{D}} \neg y$ (for $x, y \in D$).

Focusing again on the combining operator permit-overrides the $\mathcal{D}$-algebra semantics is as follows [NBL09]:

$$\begin{aligned} f_{po}(x,y) \quad = \quad & (x \oplus^{\mathcal{D}} y) \\ & \ominus^{\mathcal{D}}(((x \otimes^{\mathcal{D}} \{\, \mathbf{p} \,\}) \oplus^{\mathcal{D}} (y \otimes^{\mathcal{D}} \{\, \mathbf{p} \,\})) \odot^{\mathcal{D}} \{\, \mathbf{d}, \tfrac{\mathbf{n}}{\mathbf{a}} \,\}) \\ & \ominus^{\mathcal{D}}(\neg((x \odot^{\mathcal{D}} y) \otimes^{\mathcal{D}} \{\, \tfrac{\mathbf{n}}{\mathbf{a}} \,\}) \odot^{\mathcal{D}} \{\, \tfrac{\mathbf{n}}{\mathbf{a}} \,\} \odot^{\mathcal{D}} \neg((x \otimes^{\mathcal{D}} \emptyset) \oplus^{\mathcal{D}} (y \otimes^{\mathcal{D}} \emptyset))) \end{aligned}$$

For a concrete example, let us consider two policies P_1 and P_2 where P_1 evaluates to indeterminate permit (I_{p} in our notation and $\{\, \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \,\}$ in the $\mathcal{D}$-algebra) and where P_2 evaluates to deny (`deny` in our notation and $\{\, \mathbf{d} \,\}$ in the $\mathcal{D}$-algebra). The evaluation of $f_{po}(P_1, P_2)$ then gives $f_{po}(\{\, \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \,\}, \{\, \mathbf{d} \,\})$ which equals $\{\, \mathbf{p}, \mathbf{d} \,\}$. As mentioned above this denotes a conflict and does not correspond to any of the values considered in neither XACML 3.0 [Ris13] nor XACML 2.0 [Mos05].

We conclude that the match between $\mathcal{D}$-algebras and XACML is imperfect. It is interesting to note that the $\mathcal{D}$-algebra as well as the Belnap approach discussed above are able to model conflict – which is not part of the XACML standard.

5.4.3 Summary

For completeness, we summarise in Table 5.2 the results of all the approaches considered in the present paper. The result of the permit-overrides combining operator under Belnap logic is **ff** and in the approach of Bruns et al. , the access is denied. The result of the permit-overrides combining operator under $\mathcal{D}$-algebra is $\{\, \mathbf{p}, \mathbf{d} \,\}$ and in the approach of Ni et al. approach using $\mathcal{D}$-algebra, a conflict occurs. This may be contrasted with Bruns et al. and Ni et al. that both claim that their approaches fit with XACML 2.0 and with Ni et al. that claim that their approach fits with XACML 3.0. In our view, the XACML 2.0 standard calls for the result to be indeterminate whereas the XACML 3.0 standard calls for the result to be indeterminate deny permit. We conclude that neither Belnap logic nor $\mathcal{D}$-algebra is compliant with XACML 2.0 or XACML 3.0 whereas our formulations in terms of V_6 and pairwise policy values $\mathbf{P}$ are in agreement with XACML 3.0.

5.5 Extension of XACML

XACML is an evolving standard that changes as need arises; this is clearly shown by the evolution from XACML 2.0 into XACML 3.0. Hence further evolution is conceivable in case new combining operators need to be dealt with.

Logic	P_1	P_2	Permit-Overrides Function	Result
Belnap logic	$\top\top$	**ff**	$(\top\top \oplus^B \mathbf{ff})[\top\top \mapsto \mathbf{ff}]$	**ff**
$\mathcal{D}$-algebra	$\{ \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \}$	$\{ \mathbf{d} \}$	$f_{po}(\{ \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \}, \{ \mathbf{d} \})$	$\{ \mathbf{p}, \mathbf{d} \}$
V_6	I_{p}	$\top_{\mathsf{d}}$	$\bigoplus_{\mathsf{po}}^{V_6}(\langle I_{\mathsf{p}}, \top_{\mathsf{d}} \rangle)$	I_{dp}
P	$[0, \frac{1}{2}]$	$[1, 0]$	$\bigoplus_{\mathsf{po}}^{\mathbf{P}}(\langle [0, \frac{1}{2}], [1, 0] \rangle)$	$[\frac{1}{2}, \frac{1}{2}]$

Table 5.2: Results of the Permit-Overrides combining operator for composing two policies P_1 (Indeterminate Permit) and P_2 (Deny) under various approaches.

We believe that our formalisation of XACML provides a good starting point for discussing such extensions and will illustrate this point of view in the present section. It should be stressed that the explanations in the current section are not necessarily in agreement with the decisions that might be adopted by OASIS.

In this section we first consider an All-Permit combining operator which can easily be dealt with in our formalisation. We next consider a Consensus-Based-Vote Combining Operator that can be dealt with in our formalisation although one could argue that it would be more appropriate to extend XACML to model conflict in the manner of Belnap 4-valued logic and $\mathcal{D}$-algebra.

5.5.1 All-Permit Combining Operator

In critical systems, usually an access is granted only if all of the components agree to grant it. In other words, the access is denied even if there is just one component that does not grant it. Let us introduce the *all-permit combining operator* to capture this.

We can formalise this in V_6 as follows. Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. The *all-permit combining operator under* V_6 is defined as follows:

$$\bigoplus_{\mathtt{all\_permit}}^{V_6}(\langle s_1, \ldots, s_n \rangle) = \begin{cases} \top_{\mathsf{p}} & \forall i : s_i = \top_{\mathsf{p}} \\ \top_{\mathsf{d}} & \text{otherwise} \end{cases}$$

An equivalent formalisation can be performed in pairwise policy values **P**. Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values from **P**. The *all-permit*

combining operator under pairwise policy values $\mathbf{P}$ is defined as follows:

$$\bigoplus_{\texttt{all\_permit}}^{\mathbf{P}}(\langle s_1,\ldots,s_n\rangle) = \begin{cases} [0,1] & Max_{\sqsubseteq_{\mathbf{P}}}(\{\ s_1,\ldots,s_n\ \}) = Min_{\sqsubseteq_{\mathbf{P}}}(\{\ s_1,\ldots,s_n\ \}) \\ & = [0,1] \\ [1,0] & \text{otherwise} \end{cases}$$

It is immediate to show the equivalence of these formalisations.

5.5.2 Consensus-Based-Vote Combining Operator

A consensus decision is reached when the majority of the policies agree to grant (or deny) an access; this can be described as follows:

1. the decision is permit if the number of permit values is higher than the number of deny values,
2. the decision is deny if the number of deny values is higher than the number of permit values,
3. the decision is not applicable if all of the policies are not applicable,
4. the decision is both deny and permit if the number of deny values is equal to the number of permit values.

In our first approach we shall stay within the possibilities offered by XACML 3.0. We shall only give the formalisation in terms of the pairwise policy values $\mathbf{P}$. First we define the function *sum* over $\{\ 0, \frac{1}{2}, 1\ \}$ to count the effect values by setting

$$sum(\langle s_1,\ldots,s_n\rangle) = \sum_{i=1}^{n} s_i$$

where $\langle s_1,\ldots,s_n\rangle$ is a sequence of 0's, $\frac{1}{2}$'s or 1's, and we then extend it to pairwise policy values by setting

$$Sum_{\sqsubseteq_{\mathbf{P}}}(S) = [sum(\langle \mathsf{d}(X)|X \in S\rangle), sum(\langle \mathsf{p}(X)|X \in S\rangle)]$$

where now S is a sequence of pairwise policy values in $\mathbf{P}$.

Taking the view that an indeterminate value is worth half of that of an applicable value we then define the *consensus-based-vote combining operator* as follows

$$\bigoplus_{\texttt{consensus\_voting}}^{\mathbf{P}}(S) = \begin{cases} [1,0] & \text{if } Sum_{\sqsubseteq_{\mathbf{P}}}(S) = [D,P], D > P \\ [0,1] & \text{if } Sum_{\sqsubseteq_{\mathbf{P}}}(S) = [D,P], P > D \\ [0,0] & \text{if } Sum_{\sqsubseteq_{\mathbf{P}}}(S) = [0,0] \\ [\frac{1}{2},\frac{1}{2}] & \text{if } Sum_{\sqsubseteq_{\mathbf{P}}}(S) = [D,P], D = P \end{cases}$$

where $S = \langle s_1, \ldots, s_n \rangle$ is a sequence of pairwise policy values from $\mathbf{P}$.

Example 5.4 Suppose we have 10 policies such as 4 policies evaluate to permit, 2 policies evaluate to deny, 3 policies evaluate to indeterminate deny I_{d} value and 1 policy evaluates to not applicable. This is captured by the sequence

$$\mathbb{P} = \langle [0,1], [0,1], [0,1], [0,1], [1,0], [1,0], [\frac{1}{2}, 0], [\frac{1}{2}, 0], [\frac{1}{2}, 0], [0,0] \rangle$$

of policy values (note that in this case the order does not matter). The result of evaluating $\mathbb{P}$ with consensus-based-vote combining operator is $[0,1]$ (i.e., it returns permit value) because $Sum_{\sqsubseteq_{\mathbf{P}}}(\mathbb{P}) = [3\frac{1}{2}, 4]$ and the permit value is greater than the deny value.

Part III

A Tool

Chapter 6

Logic Programming Preliminaries

> There are fixed points throughout time where things must stay exactly the way they are. This is not one of them. This is an opportunity! Whatever happens here will create its own timeline, its own reality, a temporal tipping point. The future revolves around you, here, now, so do good!
>
> The Doctor, *Doctor Who*

Logic programming offers useful methods and techniques to software engineers. Several research and industrial projects have either successfully applied logic programming languages during the software development life cycle, or have developed useful software engineering tools exploiting some feature of logic programming. Ciancarini and Levi [CS95] showed a survey on software development tools based on logic programming.

In this chapter, we introduce general notation and terminology for logic programming. The chapter starts with the introduction of the syntax of well-formed formulae of a first order theory. Then it introduces the more specific syntax of logic programs and continues with the description of a declarative semantics for logic programs.

6.1 Logic Programs

We recall basic notation and terminology that we use throughout this thesis. It is based on [Llo84] with some extensions. We start with the introduction of the syntax of well-formed formulae of a first order theory. Then we introduce the more specific syntax of logic programs.

6.1.1 First-Order Language.

We consider an *alphabet* consisting of (finite or countably infinite) disjoint sets of variables, constants, function symbols, predicate symbols, connectives $\{$ **not**, $\wedge, \leftarrow \}$, punctuation symbols $\{$ "(", ",", ")", "." $\}$. Additionally, the alphabet also contains the special symbols $\top$ denoting a valid formula.

Next we turn to the definition of the first order language given by an alphabet.

Definition 6.1 (Term) The set of *terms* is the smallest set defined by the following rules:

1. A variable is a term.
2. A constant is a term.
3. If f is an n-ary function symbol ($n \geq 1$) and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

A *ground term* is a term not containing variables.

We will use upper case letters to denote variables and lower case letters to denote constants, function- and predicate symbols.

Definition 6.2 (Formula) The set of *(well-formed) formulae* is the smallest set defined by the following rules:

1. If p is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula (called an *atomic formula* or simply an *atom*).
2. If F and G are formulae, then so are (not F), $(F \wedge G)$, and $(F \leftarrow G)$.

A *propositional formula* is a formula where all predicate symbols are of arity 0. A *ground formula* is a formula with every term grounded.

We are now ready to define a language as follows:

Definition 6.3 (Language) A *language* $\mathcal{L}$ given by an alphabet $\mathcal{A}$ consists of the set of all formulae constructed from the symbols of $\mathcal{A}$.

6.1.2 Syntax of Logic Programs

A logic program is a declarative, relational style of programming based on first-order logic. In this section, we define the syntax of logic programs.

Definition 6.4 (Clause) A *(program) clause* is a formula of the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \text{not } B_{m+1} \wedge \cdots \wedge \text{not } B_n.$$

where $n \geq m \geq 1$, H is an atom, and each $B_i, 1 \leq i \leq n$, is either an atom or $\top$. H is called the *head* and $B_1 \wedge \cdots \wedge B_m \wedge \text{not } B_{m+1} \wedge \cdots \wedge \text{not } B_n$ the *body* of the clause. We usually write $B_1 \wedge \cdots \wedge B_m \wedge \text{not } B_{m+1} \wedge \cdots \wedge \text{not } B_n$ simply as $B_1, \cdots, B_m, \text{not } B_{m+1}, \cdots, \text{not } B_n$ and we finish each clause with a dot as in Prolog.

We define a *definite clause* as a clause where every $B_i,\ 1 \leq i \leq m$, is either an atom or $\top$.

One should observe that the body of a clause must not be empty. A *fact* is a clause of the form

$$H \leftarrow \top.$$

Definition 6.5 (General Logic Program) A *(general logic) program* Π is a finite set of clauses. A *definite (logic) program* Π as a program where every clause is definite clause.

We denote *ground*(Π) for the set of all ground instances of rules in the program Π.

We assume that each non-propositional program contains at least one constant symbol. Moreover, the language $\mathcal{L}$ underlying a program Π shall contain precisely the predicate, function and constant symbols occurring in Π, and no others.

6.2 Semantics of Logic Programs

6.2.1 Basic Terminologies

The declarative semantics of a logic program is given by a model-theoretic semantics of formulae in the underlying language. This sections discusses interpretations and models, concentrating particularly on the important class of Herbrand interpretations.

Definition 6.6 (Herbrand Universe) The *Herbrand universe* $\mathcal{U}_\mathcal{L}$ for a language $\mathcal{L}$ is the set of all ground terms that can be formed from the constants and function symbols appearing in $\mathcal{L}$. By $\mathcal{U}_\Pi$ we denote the Herbrand universe for the language underlying the program Π.

Definition 6.7 (Ground Instance) A *ground instance* of a formula F is any ground formula that results from F by substituting all variables by terms in $\mathcal{U}_\mathcal{L}$. We denote by $ground(\Pi)$ the set of all ground instances of clauses in program Π.

In many cases, $ground(\Pi)$ is infinite. In the sequel, we will consider $ground(\Pi)$ as a substitute for Π, thus ignoring unification issues.

Definition 6.8 (Herbrand Base) The *Herbrand base* $\mathcal{B}_\mathcal{L}$ for a language $\mathcal{L}$ is the set of all ground atoms that can be formed by using predicate symbols from $\mathcal{L}$ and ground terms from $\mathcal{U}_\mathcal{L}$ as arguments. By $\mathcal{B}_\Pi$ we denote the Herbrand base for the language underlying the program Π.

Since in this work we only use Herbrand interpretations, we drop the qualification "Herbrand".

Definition 6.9 (Interpretation) An *interpretation* I of a program Π is a mapping from the Herbrand base $\mathcal{B}_\Pi$ to the set of truth values: true and false ($\{\top, \bot\}$). All atoms belong to interpretation I are mapped to $\top$. All atoms which does not occur in I are mapped to $\bot$.

The truth value of arbitrary formulae under some interpretation can be determined from a truth table as usual (see Table 6.1).

Definition 6.10 (Truth Value of a Formula) The logical value of ground formulae can be derived from Table 6.1 in the usual way. A formula ϕ is

Table 6.1: Truth Values for Formulae

ϕ	ψ	not ϕ	$\phi \wedge \psi$	$\phi \leftarrow \psi$
$\top$	$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\bot$	$\top$

then *true under interpretation* I, denoted by $I(\phi) = \top$, if all its ground instances are true in I; it is *false under interpretation* I, denoted by $I(\phi) = \bot$, if there is a ground instance of ϕ that is false in I.

Definition 6.11 (Model) An interpretation I is a *model* of formula ϕ if $I(\phi) = \top$. For a program Π, we say I is a model of Π if I is a model for every clause in Π.

6.2.2 Minimal Model Semantics

The idea of the canonical model approach is that a declarative semantics for a class of logic programs can be defined by selecting one of its models as the "canonical" model. It is also called the *meaning* of the program. This model determines which answer to a given query is considered correct, i.e., a query without variables should be answer *yes* if it is true in the canonical model, and no otherwise. The canonical model is usually selected among the Herbrand models of the program.

Definition 6.12 (Ordering Among Interpretations) Let $\mathcal{I}$ be a collection of interpretations. Then an interpretation I is called *minimal* in $\mathcal{I}$ if and only if there is no interpretation J in $\mathcal{I}$ such that $J \subsetneq I$. An interpretation I is called *least* in $\mathcal{I}$ if and only if $I \subseteq J$ for any interpretation J in $\mathcal{I}$. A model M of a program Π is called minimal (respectively least) if it is minimal (respectively least) among all models of Π.

Example 6.1 (Beverage 1) Consider a situation where an agent Pretty drinks either tea or coffee. This situation can be represented by a program Π as follows.

Π_{drink} :
drink $\leftarrow$ *tea*.
drink $\leftarrow$ *coffee*.

All of possible interpretation for Π_{drink} is as follows.

$$\begin{array}{ll} I_1 = \emptyset, & I_5 = \{\ tea, coffee\ \}, \\ I_2 = \{\ drink\ \}, & I_6 = \{\ drink, tea\ \}, \\ I_3 = \{\ tea\ \}, & I_7 = \{\ drink, coffee\ \}, \text{ and} \\ I_4 = \{\ coffee\ \}, & I_8 = \{\ drink, tea, coffee\ \}. \end{array}$$

From those 8 interpretations, only 5 are models, i.e., I_1, I_2, I_6, I_7, I_8.

Every program can have many different models. For example, the models of the program Π_{drink} consists of an empty set and four nonempty sets. From those models, the empty set is a minimal model among all models of Π_{drink} and also the least model.

There is a guarantee that every definite program has a single minimal model and it is the least model. To compute the least model of a definite program, M. H. van Emden and R. A. Kowalski in [VEK76] introduced an consequence operator for logic program – an operator to express the consequences of the program when the bodies of its clauses are interpreted under the input interpretation. The operator is based on the Knaster-Tarski Operator. The least model can be obtained by applying the Knaster-Tarski Operator starting with an empty set until a fixed point is found. The proof of this claim is based on Knaster-Tarski Fixed Point Theorem and was published for the theory of logic programming area by K. R. Apt and M. H. van Emden in [AvE82].

Definition 6.13 (The Knaster-Tarski Operator) Let Π be a program and let I be an interpretation. The *Knaster-Tarski operator* is defined as follows.

$$T_\Pi(I) = \{A \mid A \leftarrow B_1, \ldots, B_m \in \mathit{ground}(\Pi) \text{ and } \{\ B_1, \ldots, B_m\ \} \subseteq I\} \quad (6.1)$$

We define $T_\Pi^0 = \emptyset$, and $T_\Pi^{i+1} = T_\Pi(T_\Pi^i),\ i \geq 0$.

Programs with negation may have several minimal Herbrand model. Unfortunately, the Knaster-Tarski operator cannot be used to compute a minimal model of a program with negation. See the example below.

Example 6.2 (Beverage 2) Let us continue our Beverage example. Consider that agent Pretty only can choose one drink. She only can drink either tea or coffee, but not both. We model this situation by adding new clauses to Π_{drink} as follows.

$$\begin{array}{ll} tea & \leftarrow \text{not } coffee. \\ coffee & \leftarrow \text{not } tea. \end{array}$$

Thus, the revised program for $\Pi drink$ is as follows.

$$
\begin{array}{ll}
\Pi_{drink}: & \\
drink & \leftarrow tea. \\
drink & \leftarrow coffee. \\
tea & \leftarrow \text{not } coffee. \\
coffee & \leftarrow \text{not } tea.
\end{array}
$$

In this situation, Π_{drink} has two minimal models, i.e.,

$$
\begin{aligned}
M_1 &= \{\, drink, tea \,\}, \\
M_2 &= \{\, drink, coffee \,\}.
\end{aligned}
$$

We show that the Knaster-Tarski Operator cannot reach fixed point for this example.

Note: $T^n_{\Pi}(I)$ denotes the n-th iteration of the Knaster-Tarski Operator applied to program Π with the input interpretation I.

$$
\begin{array}{ll}
T^1_{\Pi_{drink}}(\emptyset) & = \{\, tea, coffee \,\} \\
T^2_{\Pi_{drink}}(\emptyset) & = \{\, drink \,\} \\
T^3_{\Pi_{drink}}(\emptyset) & = \{\, tea, coffee \,\} = T^1_{\Pi_{drink}}(\emptyset) \\
T^4_{\Pi_{drink}}(\emptyset) & = \{\, drink \,\} = T^2_{\Pi_{drink}}(\emptyset) \\
 & \vdots \\
\text{For } n \geq 2: & \\
T^{2n-1}_{\Pi_{drink}}(\emptyset) & = T^1_{\Pi_{drink}}(\emptyset) \\
T^{2n}_{\Pi_{drink}}(\emptyset) & = T^2_{\Pi_{drink}}(\emptyset)
\end{array}
$$

6.2.3 Stable Model Semantics

The intuition behind stable model semantics is to treat negated atoms in a special way. Intuitively, given an interpretation I, if $a \in I$, then, a clause's body with the negative atom not a cannot be true. On the other hand, if $a \notin I$, then not a can be assumed true and we are safe to remove it from any body where it occurs. We say that the interpretation I as an assumption about which negated atoms are true and what are false.

Definition 6.14 (Reduct) Let I be an interpretation of program Π. Then, the Gelfond-Liftschitz reduct [GL88](or simply reduct) of Π with respect to an

interpretation I, denoted by Π^I, is as follows.

$$\Pi^I = \{A \leftarrow B_1, \ldots, B_m | \quad A \leftarrow B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n \in \Pi \text{ and } I(\text{not } B_{m+1}, \ldots, \text{not } B_n) = \top\}$$

One should consider that Π^I is a definite program, and thus has a least model, i.e., $lfp(T_{\Pi^I})$, the least fixed point of the Knaster-Tarski Operator applied to Π^I. If Π^I does not "contradict" I, one should expect that $lfp(\Pi^I) = I$. If this happens to be the case, then I can be regarded as being "stable".

Example 6.3 (Beverage 3) We have shown from previous example that Π_{drink} has two minimal models, i.e., $M_1 = \{ drink, tea \}$ and $M_2 = \{ drink, coffee \}$. In this example, we show that both models are the stable models for Π_{drink}.

$$\begin{array}{ll} \Pi_{drink}^{M_1}: & \\ drink & \leftarrow \top. \\ tea & \leftarrow \top. \end{array}$$

The $lfp(T_{\Pi_{drink}}) = \{ drink, tea \} = M_1$.

$$\begin{array}{ll} \Pi_{drink}^{M_2}: & \\ drink & \leftarrow \top. \\ coffee & \leftarrow \top. \end{array}$$

The $lfp(T_{\Pi_{drink}}) = \{ drink, coffee \} = M_2$.

Definition 6.15 (Answer Set) An interpretation I of program Π is an *answer set* of Π if I is the stable model of Π^I. For a program with variables, an interpretation I of Π is an answer set of Π if I is an answer set of $ground(\Pi)$.

Chapter 7

XACML Transformation into Logic Programs

Algorithm = Logic + Control

Robert Kowalski, a logician and computer scientist

In this chapter, we lay down the formal framework for transforming the XACML abstract syntax presented in Chapter 5 into logic programs. The transformation of XACML components is based on the semantics of each component explained in Section 5.2. The transformation of combining algorithms is based on the semantics of XACML combining algorithms explained in Section 5.3.

We show step by step how to transform XACML components into logic programs in Section 7.1 and in Section 7.2 we present the transformation of XACML combining algorithms.

7.1 XACML Policy Components Transformation

The evaluation of XACML components depends on the given Request element. Hence, before we move forward, we show how the transformation of Request element into a logic program. We transform all members of Request element into facts.

Request syntax: Let $\mathcal{Q} = \texttt{request:}\{\mathcal{A}t_1, \ldots, \mathcal{A}t_n\}$ be a Request component.

Request transformation: The transformation of Request $\mathcal{Q}$ into the logic program $\Pi_{\mathcal{Q}}$ is as follows.

$$\mathcal{A}t_i \quad \leftarrow \top. \;\; 1 \leq i \leq n$$

The transformation of XACML policy components is based on its semantics. We begin by showing the semantics semantics evaluation of XACML components explained in Section 5.2 then we show how the transformation into logic programs. Please note that the calligraphic font in each transformation indicates the XACML component's name, that is, it does not represent a variable in logic program.

We use a two-place function $val/2$ to indicate the semantics of XACML components where the first argument is the name of XACML component and the second argument is its value. Given a semantic equation of the form

$$[\![X]\!](\mathcal{Q}) = v \text{ if } condition_1 \text{ and } \ldots \text{ and } condition_n$$

we produce a rule of the form $val(X, v) \leftarrow condition_1, \ldots, condition_n$.

Given a semantic equation of the form

$$[\![X]\!](\mathcal{Q}) = v \text{ if } condition_1 \text{ or } \ldots \text{ or } condition_n$$

we produce a rule of the form $val(X, v) \leftarrow condition_i$. $1 \leq i \leq n$.

7.1.1 Transformation of Match Component

The evaluation of Target component starts from the very basic component, i.e., the evaluation of Match component, and it continues to the evaluation

of AnyOf component, AllOf component, and finally the evaluation of Target component. Therefore, we start from the transformation of Match component, AnyOf component, AllOf component, and finally Target component.

Match syntax: Let $\mathcal{M} = \boldsymbol{f_{MatchID}}(v,\ cat)$ be a Match component where v is an attribute value and cat is an attribute category.

evalM evaluation: The evaluation of Match component over Request element $\mathcal{Q} = \texttt{request:}\ \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is as follows.

$$\texttt{eval}_M(cat, \mathcal{Q}) = \begin{cases} \texttt{error} & \text{if an error occurs during the evaluation process} \\ \{v_1, \ldots, v_m\} & \text{if } cat(v_i) \in \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}, 1 \leq i \leq m \leq n \end{cases}$$

evalM transformation: Let $\mathcal{Q}$ be a Request component. The transformation for $\texttt{eval}_M(cat, \mathcal{Q})$ is as follows.

$$\begin{array}{ll} \Pi_{\texttt{eval}_M}: & \\ evalM(cat, \texttt{error}) & \leftarrow \texttt{error}. \\ evalM(cat, V) & \leftarrow cat(V). \end{array}$$

Match evaluation:

$$[\![f_{MatchID}(v, cat)]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \texttt{eval}_M(cat, \mathcal{Q}) \neq \texttt{error} \text{ and} \\ & \quad \exists v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \top \\ \bot & \text{if } \texttt{eval}_M(cat, \mathcal{Q}) = \{\ \} \text{ or} \\ & \quad (\texttt{eval}_M(cat, \mathcal{Q}) \neq \texttt{error} \text{ and} \\ & \quad\ \forall v' \in \texttt{eval}_M(C, \mathcal{Q}) : f_{MatchID}(v, v') = \bot) \\ I & \text{if } \texttt{eval}_M(cat, \mathcal{Q}) = \texttt{error} \text{ or} \\ & \quad ((\forall v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') \neq \top) \text{ and} \\ & \quad\ (\exists v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = I)) \end{cases}$$

Match transformation: First, we need to model the notion of empty set in logic program. The transformation of empty set w.r.t. the evaluation of Match component is as follows.

$$\begin{array}{ll} not\_emptybag(cat) & \leftarrow cat(V). \\ evalM(cat, \mathsf{emptybag}) & \leftarrow \text{not } not\_emptybag(cat). \end{array}$$

The first encoding ensures that the bag as a collector for attribute values of category *cat*in the Request element is not empty whenever the program finds the attribute value matched with the category *cat*. The second encoding models that an empty bag obtained if the program cannot find not empty bag by default.

Let $\mathcal{M} = f_{MatchID}(v, c)$ be a Match component and $\mathcal{Q}$ be a Request component. The transformation of Match $\mathcal{M}$ into logic program $\Pi_{\mathcal{M}}$ is as follows.

$\Pi_{\mathcal{M}}$:

$val(\mathcal{M}, \mathsf{m})$	$\leftarrow evalM(cat, V), V \neq \texttt{error}, f_{MatchID}(v, V, \texttt{true}).$
$val(\mathcal{M}, \mathsf{nm})$	$\leftarrow evalM(cat, \mathsf{emptybag}).$
$val(\mathcal{M}, \mathsf{nm})$	$\leftarrow evalM(cat, V), V \neq \texttt{error},$
	$\leftarrow \text{not } not\_false not\_false(f_{MatchID}(v, cat)).$
$not\_false(f_{MatchID}(v, cat))$	$\leftarrow evalM(cat, V), \text{not } f_{MatchID}(v, V, \texttt{false}).$
$val(\mathcal{M}, \mathsf{idt})$	$\leftarrow evalM(cat, \texttt{error}).$
$val(\mathcal{M}, \mathsf{idt})$	$\leftarrow evalM(cat, V), f_{MatchID}(v, V, \mathsf{idt}),$
	$\text{not } true(f_{MatchID}(v, cat)).$
$true(f_{MatchID}(v, cat))$	$\leftarrow evalM(cat, v), f_{MatchID}(v, V, \texttt{true}).$

Example 7.1 (Example of Match Transformation) The transformation for $\mathcal{M} = \mathsf{string\_equal}(\mathsf{read}, \mathsf{action})$ is as follows.

$\Pi_{\mathcal{M}}$:

$evalM(action, \texttt{error})$	$\leftarrow \texttt{error}.$
$evalM(action, V)$	$\leftarrow action(V).$
$not\_emptybag(action)$	$\leftarrow action(V).$
$evalM(action, \mathsf{emptybag})$	$\leftarrow \text{not } not\_emptybag(action).$
$val(string\_equal(read, action), \mathsf{m})$	$\leftarrow evalM(action, V), V \neq \texttt{error},$
	$string\_equal(read, V, \texttt{true}).$
$val(string\_equal(read, action), \mathsf{nm})$	$\leftarrow evalM(action, \mathsf{emptybag}).$
$val(string\_equal(read, action), \mathsf{nm})$	$\leftarrow evalM(action, V), V \neq \texttt{error},$
	$string\_equal(read, V, \texttt{false}),$
	$\text{not } not\_false(string\_equal(read, action)).$
$not\_false(string\_equal(read, action))$	$\leftarrow evalM(action, V), string\_equal(read, V, \texttt{true}).$
$not\_false(string\_equal(read, action))$	$\leftarrow evalM(action, V), string\_equal(read, V, \mathsf{idt}).$
$val(string\_equal(read, action), \mathsf{idt})$	$\leftarrow evalM(action, V), string\_equal(v, V, \mathsf{idt}),$
	$evalM(action, V'), string\_equal(v, V', \mathsf{idt})$

Example 7.2 (Example of $f_{MatchID}/2$ Transformation) Let we define string_equal as follows.

$$\mathsf{string\_equal}(X, Y) = \begin{cases} \top & \text{if } X = Y \text{ and there are no errors} \\ \bot & \text{if } X \neq Y \text{ and there are no errors} \\ I & \text{if an error occurs during the evaluation process} \end{cases}$$

The transformation for string_equal/2 is as follows.

$$
\begin{array}{ll}
string\_equal(X, X, \texttt{true}) & \leftarrow \text{not } \texttt{error}. \\
string\_equal(X, Y, \texttt{false}) & \leftarrow X \neq Y, \text{not } \texttt{error}. \\
string\_equal(X, Y, \mathsf{idt}) & \leftarrow \texttt{error}.
\end{array}
$$

7.1.2 Transformation of AnyOf, AllOf and Target Components

AllOf syntax: Let $\texttt{allof}(\mathcal{A}_{id}) : \{\, \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n \,\}$, $n \geq 1$ be an AllOf component where each $\mathcal{M}_i, 1 \leq i \leq n$, is a Match component.

AllOf semantics:

$$
[\![\mathcal{A}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{M}_i]\!] = \top \\ \bot & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{M}_i]\!] = \bot \\ I & \text{otherwise} \end{cases}
$$

AllOf transformation: The transformation of AllOf $\mathcal{A}_{id}$ into logic program $\Pi_{\mathcal{A}_{id}}$ is as follows.

$$
\begin{array}{ll}
\Pi_{\mathcal{A}_{id}} : & \\
val(\mathcal{A}_{id}, \mathsf{m}) & \leftarrow val(\mathcal{M}_1, \mathsf{m}), \ldots, val(\mathcal{M}_n, \mathsf{m}). \\
val(\mathcal{A}_{id}, \mathsf{nm}) & \leftarrow val(\mathcal{M}_i, \mathsf{nm}).\ (1 \leq i \leq n) \\
val(\mathcal{A}_{id}, \mathsf{idt}) & \leftarrow \text{not } val(\mathcal{A}_{id}, \mathsf{m}), \text{not } val(\mathcal{A}_{id}, \mathsf{nm}).
\end{array}
$$

AnyOf syntax: Let $\texttt{anyof}(\mathcal{E}_{id}) : \{\, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \,\}$, $n \geq 1$ be a AnyOf elements and each $\mathcal{A}_i$ is an AllOf element.

AnyOf semantics:

$$
[\![\mathcal{E}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{A}_i]\!] = \top \\ \bot & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{A}_i]\!] = \bot \\ I & \text{otherwise} \end{cases}
$$

AnyOf transformation: The transformation of AnyOf $\mathcal{E}_{id}$ into logic program $\Pi_{\mathcal{E}_{id}}$ is as follows.

$$
\begin{array}{ll}
\Pi_{\mathcal{E}_{id}} : & \\
val(\mathcal{E}_{id}, \mathsf{m}) & \leftarrow val(\mathcal{A}_i, \mathsf{m}).\ (1 \leq i \leq n) \\
val(\mathcal{E}_{id}, \mathsf{nm}) & \leftarrow val(\mathcal{A}_1, \mathsf{nm}), \ldots, val(\mathcal{A}_n, \mathsf{nm}). \\
val(\mathcal{E}_{id}, \mathsf{idt}) & \leftarrow \text{not } val(\mathcal{E}_{id}, \mathsf{m}), \text{not } val(\mathcal{E}_{id}, \mathsf{nm}).
\end{array}
$$

Target syntax: Let $\texttt{allof}(\mathcal{T}_{id}) : \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$, $n \geq 1$ be a Target elements and each $\mathcal{E}_i$ is an AnyOf element.

Target semantics:

$$
[\![\mathcal{T}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{E}_i]\!] = \top \\ \bot & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{E}_i]\!] = \bot \\ I & \text{otherwise} \end{cases}
$$

Target transformation: The transformation of Target $\mathcal{T}_{id}$ into logic program $\Pi_{\mathcal{T}_{id}}$ is as follows.

$$
\begin{array}{ll}
\Pi_{\mathcal{T}_{id}} : & \\
val(\mathsf{null}, \mathsf{m}) & \leftarrow \top. \\
val(\mathcal{T}_{id}, \mathsf{m}) & \leftarrow val(\mathcal{E}_1, \mathsf{m}), \ldots, val(\mathcal{E}_n, \mathsf{m}). \\
val(\mathcal{T}_{id}, \mathsf{nm}) & \leftarrow val(\mathcal{E}_i, \mathsf{nm}).\ (1 \leq i \leq n) \\
val(\mathcal{T}_{id}, \mathsf{idt}) & \leftarrow \text{not } val(\mathcal{T}_{id}, \mathsf{m}), \text{not } val(\mathcal{T}_{id}, \mathsf{nm}).
\end{array}
$$

7.1.3 Transformation of Condition Component.

The transformation of Condition $\mathcal{C}$ into logic program $\Pi_{\mathcal{C}}$ is as follows

$$val(\mathcal{C}, V) \quad \leftarrow \texttt{eval}(\mathcal{C}, V).$$

Moreover, the transformation of Condition also depends on the transformation of `eval` function into logic program. Since we do not describe specific `eval` functions, we leave this transformation to the user.

Example 7.3 (Example of Condition Transformation) Suppose we would like to transform the Condition of r1 from previous example. Recall that the Condition of r1 is as follows.

```
1 condition(r1):{ patient_number(N) /\ patient_record(patient_id,
      X) /\ N = X }
```

A possible `eval` function for "rule r1: patient only can see his or her patient record" is

$\Pi_{condition(r1)}$:

$$
\begin{array}{ll}
val(condition(r1), V) & \leftarrow evalC(condition(r1), V). \\
evalC(condition(r1), \mathsf{t}) & \leftarrow patient\_number(X), patient\_record(patient\_id, X), \\
 & \quad \text{not } \texttt{error}(patient\_number(X)), \\
 & \quad \text{not } \texttt{error}(patient\_record(patient\_id, X)). \\
evalC(condition(r1), \mathsf{f}) & \leftarrow patient\_number(X), patient\_record(patient\_id, X), \\
 & \quad X \neq Y, \text{not } \texttt{error}(patient\_number(X)), \\
 & \quad \text{not } \texttt{error}(patient\_record(patient\_id, Y)). \\
evalC(condition(r1), \mathsf{idt}) & \leftarrow \text{not } evalC(condition(r1), \mathsf{t}), \text{not } evalC(condition(r1), \mathsf{f}).
\end{array}
$$

The $\texttt{error}(patient\_number(X))$ and $\texttt{error}(patient\_record(patient\_id, X)$ indicate possible errors that might occur, e.g., the system could not connect to the database so that the system does not know the identifier of the patient.

7.1.4 Transformation of Rule Component.

Rule syntax: Let $\mathcal{R} = \texttt{rule}(R_{id}) : \{ E \ ; \ \mathcal{T} \ ; \ \mathcal{C} \}$ be a Rule component where E is the Rule's effect, $E \in \{ \texttt{permit}, \texttt{deny} \}$, $\mathcal{T}$ is a Target and $\mathcal{C}$ is a Condition.

Rule evaluation:

$$
[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \begin{cases}
\top_{\mathsf{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \text{ and } E = \texttt{deny} \\
\top_{\mathsf{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \text{ and } E = \texttt{permit} \\
I_{\mathsf{d}} & \text{if } \big(([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = I) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = I\big) \text{ and } \\
 & \quad E = \texttt{permit} \\
I_{\mathsf{p}} & \text{if } \big(([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = I) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = I\big) \text{ and } \\
 & \quad E = \texttt{deny} \\
\bot & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \bot\big) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot
\end{cases}
$$

Rule transformation: The transformation of Rule $\mathcal{R}_{id}$ into logic program $\Pi_{\mathcal{R}_{id}}$ is as follows.

$$
\begin{array}{ll}
\Pi_{\mathcal{R}_{id}}: & \\
rule(\mathcal{R}_{id}, E) & \leftarrow \top. \\
val(\mathcal{R}_{id}, E) & \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{t}). \\
val(\mathcal{R}_{id}, \mathsf{i_d}) & \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{deny}. \\
val(\mathcal{R}_{id}, \mathsf{i_d}) & \leftarrow val(\mathcal{T}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{deny}. \\
val(\mathcal{R}_{id}, \mathsf{i_p}) & \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{permit}. \\
val(\mathcal{R}_{id}, \mathsf{i_p}) & \leftarrow val(\mathcal{T}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{permit}. \\
val(\mathcal{R}_{id}, \mathsf{na}) & \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{f}). \\
val(\mathcal{R}_{id}, \mathsf{na}) & \leftarrow val(\mathcal{T}, \mathsf{nm}).
\end{array}
$$

In this transformation, the E is not a variable because we should change E based on what is written in the Rule's effect.

7.1.5 Transformation of Policy and PolicySet Components.

Policy syntax: Let $\texttt{policy}(\mathcal{P}_{id}) : \{\, \text{comb}_{id} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ be a Policy component where $\mathcal{T}$ is a Target, $\langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ is a sequence of Rule elements and comb_{id} is a combining algorithm identifier.

Policy evaluation:

$$
[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \begin{cases}
\top_{\mathsf{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \top_{\mathsf{d}} \\
\top_{\mathsf{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \top_{\mathsf{p}} \\
I_{\mathsf{dp}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = I_{\mathsf{dp}}\big) \text{ or} \\
 & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \big(I \text{ or } I_{\mathsf{dp}}\big)\big) \\
I_{\mathsf{d}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = I_{\mathsf{d}}\big) \text{ or} \\
 & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \big(\top_{\mathsf{d}} \text{ or } I_{\mathsf{d}}\big)\big) \\
I_{\mathsf{p}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = I_{\mathsf{p}}\big) \text{ or} \\
 & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \big(\top_{\mathsf{p}} \text{ or } I_{\mathsf{p}}\big)\big) \\
\bot & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\
 & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \bot\big) \text{ or} \\
 & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \bot\big)
\end{cases}
$$

where $\mathbb{R} = \langle [\![\mathcal{R}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{R}_n]\!](\mathcal{Q}) \rangle$.

Policy transformation: In order to indicate that the Policy contains Rule $\mathcal{R}_i$, for every Rule $\mathcal{R}_i \in \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$, $\Pi_{\mathcal{P}_{id}}$ contains:

$$decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, V) \quad \leftarrow val(\mathcal{R}_i, V).\ (1 \leq i \leq n)$$

The transformation for Policy $\mathcal{P}_{id}$ into logic program $\Pi_{\mathcal{P}_{id}}$ is as follows.

$\Pi_{\mathcal{P}_{id}}$:

$val(\mathcal{P}_{id}, \texttt{permit}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \texttt{permit}).$
$val(\mathcal{P}_{id}, \texttt{deny}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \texttt{deny}).$
$val(\mathcal{P}_{id}, \mathsf{i_{dp}}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_{dp}}).$
$val(\mathcal{P}_{id}, \mathsf{i_{dp}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{idt}).$
$val(\mathcal{P}_{id}, \mathsf{i_{dp}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_{dp}}).$
$val(\mathcal{P}_{id}, \mathsf{i_d}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}).$
$val(\mathcal{P}_{id}, \mathsf{i_d}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \texttt{deny}).$
$val(\mathcal{P}_{id}, \mathsf{i_d}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}).$
$val(\mathcal{P}_{id}, \mathsf{i_p}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_p}).$
$val(\mathcal{P}_{id}, \mathsf{i_p}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \texttt{permit}).$
$val(\mathcal{P}_{id}, \mathsf{i_p}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_p}).$
$val(\mathcal{P}_{id}, \mathsf{na}) \leftarrow val(\mathcal{T}, \mathsf{nm}).$
$val(\mathcal{P}_{id}, \mathsf{na}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{na}).$
$val(\mathcal{P}_{id}, \mathsf{na}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{na}).$

The transformation of PolicySet is similar to the transformation of Policy component.

PolicySet syntax: Let $\texttt{policyset}(\mathcal{PS}_{id}) : \{\, \text{comb}_{id}\ ;\ \mathcal{T}\ ;\ \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle \,\}$ be a PolicySet component where $\mathcal{T}$ is a Target, $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ is a sequence of Policy elements and comb_{id} is a combining algorithm identifier.

PolicySet evaluation:

$$[\![\mathcal{PS}_{id}]\!](\mathcal{Q}) = \begin{cases} \top_{\mathsf{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = \top_{\mathsf{d}} \\ \top_{\mathsf{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = \top_{\mathsf{p}} \\ I_{\mathsf{dp}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = I_{\mathsf{dp}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = \big(I \text{ or } I_{\mathsf{dp}}\big)\big) \\ I_{\mathsf{d}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P}) = I_{\mathsf{d}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \big(\top_{\mathsf{d}} \text{ or } I_{\mathsf{d}}\big)\big) \\ I_{\mathsf{p}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = I_{\mathsf{p}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \big(\top_{\mathsf{p}} \text{ or } I_{\mathsf{p}}\big)\big) \\ \bot & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \bot\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\mathrm{comb}_{id}}(\mathbb{P} = \bot\big) \end{cases}$$

where $\mathbb{P} = \langle [\![\mathcal{P}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{P}_n]\!](\mathcal{Q})\rangle$.

PolicySet transformation: In order to indicate that the PolicySet contains Policy $\mathcal{P}_i$, for every Policy $\mathcal{P}_i \in \langle \mathcal{P}_1, \ldots, \mathcal{P}_n\rangle$, $\Pi_{\mathcal{PS}_{id}}$ contains:

$$\mathit{decision\_of}(\mathcal{PS}_{id}, \mathcal{P}_i, V) \quad \leftarrow \mathit{val}(\mathcal{P}_i, V).\ (1 \leq i \leq n)$$

The transformation for PolicySet $\mathcal{PS}_{id}$ into logic program $\Pi_{\mathcal{PS}_{id}}$ is as follows.

$\Pi_{\mathcal{PS}_{id}}$:

$\mathit{val}(\mathcal{PS}_{id}, \mathtt{permit}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{m}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathtt{permit}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathtt{deny}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{m}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathtt{deny}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{dp}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{m}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{i}_{\mathsf{dp}}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{dp}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{idt}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{dp}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{i}_{\mathsf{dp}}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{d}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{m}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{i}_{\mathsf{d}}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{d}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathtt{deny}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{d}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{i}_{\mathsf{d}}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{p}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{m}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{i}_{\mathsf{p}}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{p}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathtt{permit}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{i}_{\mathsf{p}}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{i}_{\mathsf{p}}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{na}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{nm}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{na}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{m}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{na}).$
$\mathit{val}(\mathcal{PS}_{id}, \mathsf{na}) \leftarrow \mathit{val}(\mathcal{T}, \mathsf{idt}), \mathit{algo}(\mathrm{comb}_{id}, \mathcal{PS}_{id}, \mathsf{na}).$

7.2 Combining Algorithms Transformation

We define generic logic programs for permit-overrides combining algorithm and only-one-applicable combining algorithm. Therefore, we use a variable P to indicate a variable over Policy identifier and R, R_1 and R_2 to indicate variables over Rule identifiers. In case the evaluation of PolicySet, the input P is for PolicySet identifier, R, R_1 and R_2 are for Policy (or PolicySet) identifiers.

7.2.1 Permit-Overrides Transformation.

Permit-Overrides evaluation: Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6 and let $\mathbf{S} = \{ s_1, \ldots, s_n \}$ be a set of policy values from $\mathbb{S}$. We define the *permit-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathsf{po}}^{V_6}(\mathbb{S}) = \bigsqcup_{\mathsf{po}} \mathbf{S}$$

Permit-Overrides transformation: Let Π_{po} be a logic program obtained by permit-overrides combining algorithm transformation for the permit-overrides combining algorithm semantics). Π_{po} contains:

Π_{po} :

$$
\begin{array}{ll}
algo(\mathsf{po}, P, \mathtt{permit}) & \leftarrow decision\_of(P, R, \mathtt{permit}). \\
algo(\mathsf{po}, P, \mathsf{i_{dp}}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), decision\_of(P, R, \mathsf{i_{dp}}). \\
algo(\mathsf{po}, P, \mathsf{i_{dp}}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), \\
 & \quad decision\_of(P, R_1, \mathsf{i_p}), decision\_of(P, R_2, \mathtt{deny}). \\
algo(\mathsf{po}, P, \mathsf{i_{dp}}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), \\
 & \quad decision\_of(P, R_1, \mathsf{i_p}), decision\_of(P, R_2, \mathsf{i_d}). \\
algo(\mathsf{po}, P, \mathsf{i_p}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), \text{not } algo(\mathsf{po}, P, \mathsf{i_{dp}}), \\
 & \quad decision\_of(P, R, \mathsf{i_p}). \\
algo(\mathsf{po}, P, \mathtt{deny}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), \text{not } algo(\mathsf{po}, P, \mathsf{i_{dp}}), \\
 & \quad \text{not } algo(\mathsf{po}, P, \mathsf{i_p}), decision\_of(P, R, \mathtt{deny}). \\
algo(\mathsf{po}, P, \mathsf{i_d}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), \text{not } algo(\mathsf{po}, P, \mathsf{i_{dp}}), \\
 & \quad \text{not } algo(\mathsf{po}, P, \mathsf{i_p}), \text{not } algo(\mathsf{po}, P, \mathtt{deny}), \\
 & \quad decision\_of(P, R, \mathsf{i_d}). \\
algo(\mathsf{po}, P, \mathsf{na}) & \leftarrow \text{not } algo(\mathsf{po}, P, \mathtt{permit}), \text{not } algo(\mathsf{po}, P, \mathsf{i_{dp}}), \\
 & \quad \text{not } algo(\mathsf{po}, P, \mathsf{i_p}), \text{not } algo(\mathsf{po}, P, \mathtt{deny}), \\
 & \quad \text{not } algo(\mathsf{po}, P, \mathsf{i_d}).
\end{array}
$$

7.2.2 Legacy Permit-Overrides Transformation

Legacy Permit-Overrides evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *legacy permit-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{lpo}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathsf{dp}} & \text{if } \bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) \in \{\, I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}} \,\} \\ \bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) & \text{otherwise} \end{cases}$$

Legacy Permit-Overrides transformation: Let $\Pi_{\mathtt{lpo}}$ be a logic program obtained by permit-overrides combining algorithm transformation. $\Pi_{\mathtt{lpo}}$ contains:

$$\begin{array}{ll} \mathit{algo}(\mathtt{lpo}, P, \mathsf{i_{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{i_p}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{i_{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{i_d}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{i_{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{i_{dp}}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathtt{permit}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathtt{permit}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathtt{deny}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathtt{deny}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{na}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{na}) \end{array}$$

Let we define a new predicate $\mathit{idt\_value}/1$ to characterise all indeterminate values, i.e., indeterminate permit, indeterminate deny, and indeterminate deny permit.

$$\begin{array}{ll} \mathit{idt\_value}(\mathsf{i_d}) & \leftarrow \top. \\ \mathit{idt\_value}(\mathsf{i_p}) & \leftarrow \top. \\ \mathit{idt\_value}(\mathsf{i_{dp}}) & \leftarrow \top. \end{array}$$

Thus, we can simplify the logic program $\Pi_{\mathtt{lpo}}$ as follows.

$$\begin{array}{ll} \Pi_{\mathtt{lpo}}: & \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{i_{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, V), \mathit{idt\_value}(V). \\ \mathit{algo}(\mathtt{lpo}, P, V) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i_{dp}}), \mathit{algo}(\mathtt{po}, P, V) \end{array}$$

7.2.3 Deny-Overrides Transformation.

Deny-Overrides evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6 and let $\mathbf{S} = \{\, s_1, \ldots, s_n \,\}$ be a set of policy values from $\mathbb{S}$. We

define the *deny-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{do}}^{V_6}(\mathbb{S}) = \bigsqcup_{\mathtt{do}} \mathbf{S}$$

Deny-Overrides transformation: Let $\Pi_{\mathtt{do}}$ be a logic program obtained by permit-overrides combining algorithm transformation. $\Pi_{\mathtt{do}}$ contains:

$$\begin{array}{ll}
\mathit{algo}(\mathtt{do}, P, \mathtt{deny}) & \leftarrow \mathit{decision_of}(P, R, \mathtt{deny}).\\
\mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}), \mathit{decision_of}(P, R, \mathsf{i}_{\mathtt{dp}}).\\
\mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}), \mathit{decision_of}(P, R_1, \mathsf{i}_{\mathtt{d}}),\\
 & \quad \mathit{decision_of}(P, R_2, \mathtt{permit}).\\
\mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}),\\
 & \quad \mathit{decision_of}(P, R_1, \mathsf{i}_{\mathtt{p}}), \mathit{decision_of}(P, R_2, \mathsf{i}_{\mathtt{d}}).\\
\mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{d}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}), \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}),\\
 & \quad \mathit{decision_of}(P, R, \mathsf{i}_{\mathtt{d}}).\\
\mathit{algo}(\mathtt{do}, P, \mathtt{permit}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}), \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}),\\
 & \quad \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{d}}), \mathit{decision_of}(P, R, \mathtt{permit}).\\
\mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{p}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}), \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}),\\
 & \quad \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{d}}), \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{d}}),\\
 & \quad \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{permit}), \mathit{decision_of}(P, R, \mathsf{i}_{\mathtt{p}}).\\
\mathit{algo}(\mathtt{do}, P, \mathsf{na}) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{deny}), \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}),\\
 & \quad \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{d}}), \text{not } \mathit{algo}(\mathtt{do}, P, \mathtt{permit}),\\
 & \quad \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{p}}).
\end{array}$$

7.2.4 Legacy Deny-Overrides Transformation

Legacy Deny-Overrides evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *legacy deny-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{ldo}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathtt{dp}} & \text{if } \bigoplus_{\mathtt{do}}^{V_6}(\mathbb{S}) \in \{\, I_{\mathtt{d}}, I_{\mathtt{p}}, I_{\mathtt{dp}} \,\} \\ \bigoplus_{\mathtt{do}}^{V_6}(\mathbb{S}) & \text{otherwise} \end{cases}$$

Legacy Deny-Overrides transformation: Let $\Pi_{\mathtt{ldo}}$ be a logic program obtained by permit-overrides combining algorithm transformation. $\Pi_{\mathtt{ldo}}$ contains:

$$\begin{array}{ll}
\mathit{algo}(\mathtt{ldo}, P, \mathsf{i}_{\mathtt{dp}}) & \leftarrow \mathit{algo}(\mathtt{do}, P, V), \mathit{idt_value}(V).\\
\mathit{algo}(\mathtt{ldo}, P, V) & \leftarrow \text{not } \mathit{algo}(\mathtt{do}, P, \mathsf{i}_{\mathtt{dp}}), \mathit{algo}(\mathtt{do}, P, V)
\end{array}$$

7.2.5 Deny-Unless-Permit Transformation

Deny-Unless-Permit evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *deny-unless-permit combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{dup}}^{V_6}(\mathbb{S}) \begin{cases} \top_{\mathtt{p}} & \text{if } \exists i \in \{1, \ldots, n\} : s_i = \top_{\mathtt{p}} \\ \top_{\mathtt{d}} & \text{otherwise} \end{cases}$$

Deny-Unless-Permit transformation: Let $\Pi_{\mathtt{dup}}$ be a logic program obtained by deny-unless-permit combining algorithm transformation. $\Pi_{\mathtt{dup}}$ contains:

$$\begin{array}{ll} \Pi_{\mathtt{dup}} : \mathit{algo}(\mathtt{dup}, P, \mathtt{permit}) & \leftarrow \mathit{decision\_of}(P, R, \mathtt{permit}). \\ \mathit{algo}(\mathtt{dup}, P, \mathtt{deny}) & \leftarrow \text{not } \mathit{algo}(\mathtt{dup}, P, \mathtt{permit}). \end{array}$$

7.2.6 Permit-Unless-Deny Transformation

Permit-Unless-Deny evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *deny-unless-permit combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{pud}}^{V_6}(\mathbb{S}) = \begin{cases} \top_{\mathtt{d}} & \text{if } \exists i \in \{1, \ldots, n\} : s_i = \top_{\mathtt{d}} \\ \top_{\mathtt{p}} & \text{otherwise} \end{cases}$$

Permit-Unless-Deny transformation: Let $\Pi_{\mathtt{pud}}$ be a logic program obtained by deny-unless-permit combining algorithm transformation. $\Pi_{\mathtt{pud}}$ contains:

$$\begin{array}{ll} \Pi_{\mathtt{pud}} : & \\ \mathit{algo}(\mathtt{pud}, P, \mathtt{deny}) & \leftarrow \mathit{decision\_of}(P, R, \mathtt{deny}). \\ \mathit{algo}(\mathtt{pud}, P, \mathtt{permit}) & \leftarrow \text{not } \mathit{algo}(\mathtt{dup}, P, \mathtt{deny}). \end{array}$$

7.2.7 First-Applicable Transformation.

First-Applicable evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *first-applicable combining algorithm under* V_6 as

follows:

$$\bigoplus_{\mathtt{fa}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathtt{dp}} & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ I_{\mathtt{d}}, I_{\mathtt{p}}, I_{\mathtt{dp}} \} \wedge \forall j : (j < i) \Rightarrow (s_j = \bot) \\ s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ \top_{\mathtt{p}}, \top_{\mathtt{d}} \} \wedge \forall j : (j < i) \Rightarrow (s_j = \bot) \\ \bot & \text{otherwise} \end{cases}$$

Let $\Pi_{\mathtt{fa}}$ be a logic program obtained by first-applicable combining algorithm transformation. For each Policy (or PolicySet) that uses this combining algorithm, $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{ \mathcal{T} \ ; \ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \ ; \ \mathtt{fa} \}$, $\Pi_{\mathcal{P}_{id}}$ contains:

$$\begin{array}{ll} \Pi_{\mathtt{fa}} : & \\ \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) & \leftarrow \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_1, V), V \neq \mathtt{na}. \\ \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) & \leftarrow \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_1, \mathtt{na}), \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_2, V), V \neq \mathtt{na}. \\ & \vdots \\ \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) & \leftarrow \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_1, \mathtt{na}), \ldots, \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_{n-1}, \mathtt{na}), \\ & \quad \mathit{decision\_of}(\mathcal{P}_{id}, R_n, V). \end{array}$$

7.2.8 Only-One-Applicable Transformation.

Only-One-Applicable evaluation: Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *only-one-applicable combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{ooa}}^{V_6}(\mathbb{S}) = \begin{cases} s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ \top_{\mathtt{d}}, \top_{\mathtt{p}} \} \wedge \\ & \quad \forall j \in \{1,\ldots,n\} : j \neq i \Rightarrow s_j = \bot \\ I_{\mathtt{dp}} & \text{if } \big(\exists i \in \{1,\ldots,n\} : s_i \in \{ I_{\mathtt{d}}, I_{\mathtt{p}}, I_{\mathtt{dp}} \}\big) \vee \\ & \quad \big(\exists i, j \in \{1,\ldots,n\} : i \neq j \wedge s_i, s_j \in \{ \top_{\mathtt{d}}, \top_{\mathtt{p}} \}\big) \\ \bot & \text{otherwise} \end{cases}$$

Only-One-Application transformation: Let we define a new predicate $\mathit{applicable\_value}/1$ to characterise all applicable values, i.e., deny and permit.

$$\begin{array}{ll} \mathit{applicable\_value}(\mathtt{deny}) & \leftarrow \top. \\ \mathit{applicable\_value}(\mathtt{permit}) & \leftarrow \top. \end{array}$$

Let $\Pi_{\mathtt{ooa}}$ be a logic program obtained by only-one-applicable combining algo-

rithm transformation. Π_{ooa} contains:

Π_{ooa} :

$$
\begin{array}{ll}
algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) & \leftarrow decision\_of(\mathcal{P}_{id}, \mathcal{R}_1, V_1), decision\_of(\mathcal{P}_{id}, \mathcal{R}_2, V_2), \mathcal{R}_1 \neq \mathcal{R}_2, \\
& \quad applicable\_value(V_1), applicable\_value(V_2). \\
algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) & \leftarrow decision\_of(\mathcal{P}_{id}, \mathcal{R}, V), idt\_value(V). \\
algo(\mathsf{ooa}, \mathcal{P}_{id}, V) & \leftarrow \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}), \\
& \quad decision\_of(\mathcal{P}_{id}, \mathcal{R}, applicable\_value), applicable\_value(V). \\
algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{na}) & \leftarrow \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}), \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathtt{deny}), \\
& \quad \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathtt{permit}).
\end{array}
$$

7.3 Semantics Relation between XACML and $\Pi_{\mathbf{XACML}}$

We have discussed in Section 6.2 that in general, logic programs may have none, one, or many answer sets. Definite logic programs have a guarantee that always have a unique answer set, but it is not always the case for general logic programs. However, acyclic programs are guaranteed to have a unique answer set [Bar03].

We say that a program is *acyclic* when there is no cycle in the program. The acyclicity in the program is guaranteed by the existence of a certain fixed assignment of natural numbers to atoms that is called a *level mapping*.

Definition 7.1 (Level Mapping) A *level mapping* for a program Π is a function

$$\lambda : \mathcal{B}_\Pi \rightarrow \mathbf{N}$$

where $\mathbf{N}$ is the set of natural numbers and $\mathcal{B}_\Pi$ is the Herbrand base for Π. We extend the definition of level mapping to a mapping from ground literals to natural numbers by setting $\lambda(\text{not } A) = \lambda(A)$.

Definition 7.2 (Acyclic Programs) Let Π be a logic program and l be a level mapping for Π. Π is *acyclic with respect to* l if for every clause $A \leftarrow B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n$ in *ground*(Π) we find

$$\lambda(A) > \lambda(B_i) \quad \text{for all } i \text{ with } 1 \leq i \leq n$$

Π is *acyclic* if it is acyclic with respect to some degree of level mapping.

Example 7.4 Suppose we have program Π_1 as following.

$$
\begin{array}{ll}
a & \leftarrow b. \\
b & \leftarrow a.
\end{array}
$$

Program Π_1 is cyclic since we cannot fulfil $\lambda(a) > \lambda(b)$ and $\lambda(b) > \lambda(a)$ in the same time. However program Π_2 is definite program. Hence, there is a unique answer set, i.e., $\emptyset$.

Suppose we have program Π_2 as following.

$$\begin{array}{ll} a & \leftarrow \text{not } b. \\ b & \leftarrow \text{not } a. \end{array}$$

For the same reason as program Π_1, program Π_2 is also cyclic. There are two answer sets for Π_2, i.e., $\{ a \}$ and $\{ b \}$.

We can see from Section 7.1 and Section 7.2 that all of the XACML 3.0 transformation programs are acyclic. Thus, it is guaranteed that Π_{XACML} has a unique answer set.

Proposition 7.3 *Let Π_{XACML} be a program obtained from XACML 3.0 element transformations and let $\Pi_{\mathcal{Q}}$ be a program transformation of Request $\mathcal{Q}$. Let M be the answer set of $\Pi_{XACML} \cup \Pi_{\mathcal{Q}}$. Then the following equation holds*

$$[\![X]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(X, V) \in M \ .$$

where X is an XACML component.

The transformation of each component into a logic program is based on exactly the definition of its XACML evaluation. The proof of this proposition can be seen in Appendix B.

7.4 Related Work

Our approach is inspired by the work of Ahn *et al.* [AHLM10a, AHLM10b]. There are three main differences between our approach and the work of Ahn *et al.*

First, while they consider XACML version 2.0 [Mos05], we address the newer version, XACML 3.0. The main difference between XACML 3.0 and XACML 2.0 is the treatment of indeterminate values. As a consequence, the combining algorithms in XACML 3.0 are more complex than the ones in XACML 2.0. XACML 2.0 only has a single indeterminate value while XACML 3.0 distinguishes between the following three types of indeterminate values:

i. *Indeterminate permit* (I_{p}) – an indeterminate value arising from a policy which could have been evaluated to permit but not deny;

ii. *Indeterminate deny* (I_{d}) – an indeterminate value arising from a policy which could have been evaluated to deny but not permit;

iii. *Indeterminate deny permit* (I_{dp}) – an indeterminate value arising from a policy which could have been evaluated as both deny and permit.

Second, Ahn *et al.* produce a monolithic logic program that can be used for the analysis of XACML policies while we take a more modular approach by first modelling an XACML Policy Decision Point as a logic program and then using this encoding within a larger program for property analysis. While Ahn, *et al.* only emphasize the "indeterminate" value in the combining algorithms, we deal with the "indeterminate" value in all XACML components, i.e., in Match, AnyOf, AllOf, Target, Condition, Rule, Policy and PolicySet components.

Finally, Ahn *et al.* translate the XACML specification directly into logic programming, so the ambiguities in the natural language specification of XACML are also reflected in their encodings. To avoid this, we base our encodings on our formalisation of XACML from Chapter 5.

Part IV

Application

Chapter 8

Access Control Property Analysis

"Logic is everywhere ... "

Prof. Steffen Hölldobler – International Center for Computational Logic, TUD, Dresden

Answer Set Programming (ASP) is a declarative programming language which one of its roots is Prolog (Programming in Logic). Although, Prolog grew out of programming with Horn clauses – a subset of first order logic, several non declarative features were included in Prolog in order to make it programmer friendly. The Prolog systems is viewed as a top-down query answering systems that the positioning of literals in the body of a clause and the ordering of clauses in the program matters in Prolog. From the perspective of ASP, a program is a set of ASP clauses and each ASP body of an ASP clause is a set of literals. Thus, the ordering of clauses and the positioning of literals in a clause do not matter in ASP.

ASP is completely different from traditional procedural programming paradigm. Instead of writing algorithms to solve a problem in hand, the programmer describes the problem using a formal language and an underlying engine finds a

solution to the problem. We illustrate the way how ASP solves problems in Figure 8.1.

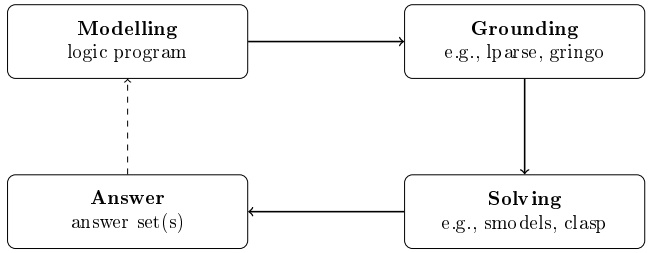

Figure 8.1: ASP System

The first step is modelling phase. The result of this phase is a representation of the given problem in terms of logic program clauses. Usually, the resulting program is formulated in firs-order variables. Thus, in the next phase, the grounding phase, all the first-order variables are systematically replaced by elements in the Herbrand Universe. This yields a finite propositional program that is then fed into the actual ASP solver. The output of the solver often consists of a textual representation of a sequence of answer sets. The modelling phase can be repeated again to refine the last version of the problem representation.

ASP has its roots in Knowledge Representation and (Nonmonotonic) Reasoning, Logic Programming (with negation), Database, and Boolean Constraint Solving. ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way, offering more succinct problem representations than propositional logic [LR06]. Meanwhile, ASP has been used in many application areas, among them, product configuration [SN98], decision support for NASA shuttle controllers [NBG+01], composition of Renaissance music [BBDVf08, BBDvF11], synthesis of multiprocessor system [IMB+09], reasoning tools in system biology [ET08, GST+08], team-building [GIL+10], and many more. The success story of ASP has its roots in the early availability of ASP solvers, beginning with the smodels system[1], followed by dlv[2], SAT-based ASP solvers, like assat[3] and cmodels[4], and the conflict-driven ASP solver clasp[5].

Implementation. The transformation of XACML components and the analysis of access control properties in this chapter have been implemented using:

[1] `http://www.tcs.hut.fi/Software/smodels/`
[2] `http://www.dlvsystem.com/`
[3] `http://assat.cs.ust.hk/`
[4] `http://www.cs.utexas.edu/users/tag/cmodels/`
[5] `http://potassco.sourceforge.net/`

- ANTLR[6] for the parser generator from the grammar presented in Chapter 4,
- Java[7] for transforming XACML components into LPs presented in Chapter 7 automatically, and
- clasp as the ASP solver.

The implementation is available at `https://sites.google.com/site/xacmlasp/tool`.

In this chapter we show how to use ASP for analysing access control security properties through Π_{XACML}. In most cases, ASP solver can solve combinatorial problems efficiently [DFP09, ABGG12]. There are several combinatorial problems in analysis access control policies, e.g., gap-free property and conflict-free property[8] [SD01, BH08].

Section 8.1 presents additional background that we will need later. Section 8.2 presents a mechanism to generate all possible queries which we will use for analysing access control properties in Section 8.3. Section 8.4 and Section 8.5 present analysis on incompleteness policies (gap-free analysis) and analysis on conflict, respectively. We present irrelevant policies analysis in Section 8.6.

8.1 Preliminaries

8.1.1 Integrity Constraints

Integrity constraints play an important role in database. Integrity constraints provide a powerful and simple to use constraint programming technique for pruning unwanted stable models as they cannot introduce new stable models but only can eliminate them.

Definition 8.1 (Constraint) An *(integrity) constraint* is a clause in the form

$$\bot \leftarrow B_1 \wedge \cdots \wedge B_m \quad (m \geq 1)$$

[6] `http://www.antlr.org/`

[7] `https://www.java.com/`

[8] A conflict decision never occurs when we strictly use the standard combining algorithms defined in XACML 3.0, since every combining algorithm always returns one value.

Theorem 8.2 *For any program* Π *and formula* F*, a set of* M *of atoms is an answer set for* $\Pi \cup \{ \bot \leftarrow F \}$ *iff* M *is an answer set for* Π *and* $M(F) = \bot$.

The proof can be seen in [LTT99].

For any program Π and formula F, the answer sets for the program

$$\Pi \cup \{ \bot \leftarrow \text{not } F. \}$$

are the sets of atoms that $M(F) = \top$. This fact gives a reduction of satisfiability problem for propositional formulae to the problem of computing answer sets.

Example 8.1 (Beverage 4) This example is a continuation from Example 6.3. Consider a situation where agent Pretty drinks either tea or coffee (but not both). She does not expect to have coffee if she has high blood pressure. We model this situation as follows..

Π_{drink} :

drink	$\leftarrow$ *tea*.
drink	$\leftarrow$ *coffee*.
tea	$\leftarrow$ not *coffee*.
coffee	$\leftarrow$ not *tea*.
$\bot$	$\leftarrow$ *coffee*, *high_blood_pressure*.

We add an integrity constraint at the end of the program. By adding this integrity constraint, all models that have both *coffee* and *high_blood_pressure* will be eliminated. Thus, whenever agent Pretty knows that she has high blood pressure, she will have coffee as her option.

8.1.2 Cardinality Constraints

In answer set programming, a problem is solved by devising a logic program such that the stable models of the program provide the answers to the problem and by using an implementation of the stable model semantics to compute an answer, i.e., a stable model of the resulting program. General logic program provides a framework where a variety of combinatorial, constraint satisfaction, and planning problems can be handled (see [Nie99] as reference.) These problems are typically related to choices with cardinality, costs and resources.

Consider the Beverage Example (Example 6.2). We model the choice of agent Pretty by modelling the situation as follows.

tea	$\leftarrow$ not *coffee*.
coffee	$\leftarrow$ not *tea*.

Consider that there are more available beverages, such as, soft drink, juice, water. The situation still the same that agent Pretty only can pick one beverage for her. Thus, we modify our program as follows.

$$
\begin{array}{ll}
\mathit{tea} & \leftarrow \text{not } \mathit{coffee}, \text{not } \mathit{softdrink}, \text{not } \mathit{juice}, \text{not } \mathit{water}. \\
\mathit{coffee} & \leftarrow \text{not } \mathit{tea}, \text{not } \mathit{softdrink}, \text{not } \mathit{juice}, \text{not } \mathit{water}. \\
\mathit{softdrink} & \leftarrow \text{not } \mathit{coffee}, \text{not } \mathit{tea}, \text{not } \mathit{juice}, \text{not } \mathit{water}. \\
\mathit{juice} & \leftarrow \text{not } \mathit{coffee}, \text{not } \mathit{tea}, \text{not } \mathit{softdrink}, \text{not } \mathit{water}. \\
\mathit{water} & \leftarrow \text{not } \mathit{coffee}, \text{not } \mathit{tea}, \text{not } \mathit{softdrink}, \text{not } \mathit{juice}.
\end{array}
$$

The general logic programs can represent the combinatorial problem, however it is not user friendly. In order to avoid this, we introduce *cardinality constraint* [NSS99,NS00]. The idea is that such a constraint is satisfied by a model for which the cardinality of the subset of the literals satisfied by the model is between the integers lower and upper bounds.

Definition 8.3 (Cardinality Constraint) A *cardinality constraint* C is an expression in the form

$$L \{ a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n \} U \tag{8.1}$$

where L and U are two integers giving the *lower* and *upper bound* of the constraint, respectively. In the case a missing lower bound is to be taken as $-\infty$ and an upper bound as ∞.

We denote $\mathit{lit}(C)$ for the corresponding set of literals $\{ a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n \}$.

A *cardinality constraint clause* is a clause in the form

$$C_0 \leftarrow C_1, \ldots, C_n. \tag{8.2}$$

where C_0 is either a cardinality constraint or $\bot$ and each $C_i, 1 \leq i \leq n$, is either a cardinality constraint or $\top$. We allow integrity constraint by setting the head of the clause with $\bot$.

Note that cardinality constraint clause can be seen as a generalisation of general logic clause which are a special case with cardinality constrains of the form $1 \{ L \} 1$ where L is a literal. For example, a clause $\{ a \leftarrow \text{not } b. \}$ can be written as $\{ 1 \{ a \} 1 \leftarrow 1 \{ \text{not } b \} 1. \}$.

Definition 8.4 A set of atoms S *satisfies* a cardinality constraint C of the form $L \{ a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n \} U$, denoted by $S(C) = \top$ iff $L \leq W(C, S) \leq U$ where

$$W(C, S) = | \{ l =\in \mathit{lit}(S) : S(l) = \top \} |$$

is the number of literals in C satisfied by S.

A clause $C_0 \leftarrow C_1, \ldots, C_n.$ is satisfied by S iff S satisfies C_0 whenever it satisfies each of $C_1, \ldots, C_n$.

The Beverage program can be represented using cardinal constraint clauses as follows.

$$1\,\{\, \mathit{tea}, \mathit{coffee}, \mathit{softdrink}, \mathit{juice}, \mathit{water}.\,\}\,1 \quad \leftarrow \top.$$

It is also useful to allow both *local* and *global variables* in a clause. The scope of a local variable is one constraint, whereas the scope of a global variable is the whole rule. Here is the example of vertex colouring problem.

$$1\,\{\, \mathit{coloured}(V, C)\,\}\,1 : \mathit{colour}(C)1 \leftarrow \mathit{vertex}(X).$$
$$\bot \leftarrow \mathit{edge}(V, U), \mathit{coloured}(V, C), \mathit{coloured}(U, C).$$

In this example, V is a global variable in the first clause stating the requirement that for each vertex v exactly one instance of $\mathit{coloured}(v, c)$ should be chosen such that $\mathit{colour}(c)$ holds for the term c. The set of facts $\mathit{colour}(c)$ provides the available colours.

8.2 Query Generator

In order to analyse access control property, sometimes we need to inspect all possible queries that might occur.

We use cardinality constraint to generate all possible values restored in the database for each attribute.

We encode $1\{\mathit{category}(X) : \mathit{category\_db}(X)\}1 \leftarrow \top.$ to generate one and only one attribute value for category *category* which obtained from its database. This is sufficient since we only consider only one request might occur. For example, we have the following generator:

$\mathcal{P}_{generator}$:

(1) $1\{\mathit{subject}(X) : \mathit{subject\_db}(X)\}1 \quad \leftarrow \top.$
(2) $1\{\mathit{action}(X) : \mathit{action\_db}(X)\}1 \quad \leftarrow \top.$
(3) $1\{\mathit{resource}(X) : \mathit{resource\_db}(X)\}1 \quad \leftarrow \top.$
(4) $1\{\mathit{environment}(X) : \mathit{environment\_db}(X)\}1 \quad \leftarrow \top.$

The first line of the encoding means that we only consider one and only one *subject* attribute value obtained from the subject database. The rest of the encoding means the same as the *subject* attribute.

Beside we generate attribute values, we might generate external condition that might happen. For example, we add our previous example following.

$$\begin{array}{l}\Pi_{generator}: \\ (5)\ \ 1\{age(X) : possible\_age(X)\}1 \quad \leftarrow \top.\end{array}$$

8.3 Property Analysis

The problem of verifying a security property Φ on XACML policies is not only to show that the property Φ holds on Π_{XACML} but also that we want to see the witnesses whenever the property Φ does not hold in order to help the policy developer refine the policies. Thus, we can see this problem as finding models for $\Pi_{\text{XACML}} \cup \Pi_{generator} \cup \Pi_{\neg\Phi}$. The founded model is the witness that the XACML policies cannot satisfy the property Φ.

Example 8.2 Suppose we have a security property:

Φ: An anonymous person **cannot** read any patient records.

Thus, the negation of property Φ is as follows

$\neg\Phi$: An anonymous person **can** read any patient records.

We define that anonymous persons are those who are neither patients, nor guardians, nor doctors, nor nurses. We encode $\mathcal{P}_{\neg\Phi}$ as follows

$$\begin{array}{ll}(1)\ anonymous & \leftarrow \text{not } subject(patient), \text{not } subject(guardian), \\ & \quad \text{not } subject(doctor), \text{not } subject(nurse). \\ (2)\ \bot & \leftarrow \text{not } anonymous. \\ (3)\ action(read) & \leftarrow \top. \\ (4)\ resource(patient\_record) & \leftarrow \top. \\ (5)\ \bot & \leftarrow \text{not } val(PS_{hospital}, \mathtt{p}).\end{array}$$

We list all of the requirements (lines $1-4$). We force the program to find an anonymous person (line 2). Later we force that the returned decision should be to permit (line 5). When the program $\Pi_{\text{XACML}} \cup \Pi_{generator} \cup \Pi_{\neg\Phi}$ returns

models, we conclude that the property Φ does not hold and the returned models are the flaws in the policies. On the other hand, we conclude that the property Φ is satisfied if no model is found.

The implementation of ASP for security analysis can be used more broader not only for analysis security properties. The idea is that all information is encoded as a set of facts. The property Φ that we want to check is encoded as an integrity constraint

$$\bot \leftarrow \text{not } \Phi.$$

By doing so, we force ASP to find models that satisfy Φ.

Here are some examples.

1. Answering question.
   For example: "What can user x access?".

$$\begin{array}{ll} \mathit{subject}(x) & \leftarrow \top. \\ \bot & \leftarrow \text{not } \mathit{val}(\mathcal{PS}_{root}, \mathsf{p}). \end{array}$$

2. Finding AC policies.
   For example, "Find all AC policies regarding *patient_record*".

$$\begin{array}{ll} \mathit{resource}(\mathit{patient\_record}) & \leftarrow \top. \\ \bot & \leftarrow \text{not } \mathit{val}(\mathcal{PS}_{root}, \mathsf{p}). \\ \bot & \leftarrow \text{not } \mathit{val}(\mathcal{PS}_{root}, \mathsf{d}). \end{array}$$

8.4 Analysis on Incompleteness Policies

A set of policies is *gap-free* if there is no access request for which there is an absence of decision. XACML defines that there is one PolicySet as the root of a set of policies. Hence, we say that there is a gap whenever we can find a request that makes the semantics of the $\mathcal{PS}_{root}$ is assigned to na. Formally, we define a gap as follows

$$\textbf{gap}: \exists \mathcal{Q} : [\![\mathcal{PS}_{root}]\!](\mathcal{Q}) = \mathsf{na}. \tag{8.3}$$

We force ASP solver to find the gap by the following encoding.

$$\begin{array}{ll} \Pi_{gap}: & \\ \mathit{gap} & \leftarrow \mathit{val}(\mathcal{PS}_{root}, \mathsf{na}). \\ \bot & \leftarrow \text{not } \mathit{gap}. \end{array}$$

In order to make sure that a set of policies is gap-free we should generate all possible requests and test whether at least one request is not captured by the set of policies. Thus, the answer sets of program $\mathcal{P} = \Pi_{\text{XACML}} \cup \Pi_{generator} \cup \Pi_{gap}$ are witnesses that the set of policies encoded in Π_{XACML} is incomplete. When there is no model that satisfies the program then we are sure that the set of policies captures all of possible cases.

8.5 Analysis on Conflicting Policies

We define a *conflicted policy* as a policy which has two different decision, i.e., deny and permit, in the same time. A conflict never occurs in XACML because the structure of policies where there is only one PolicySet as the root of all of policies and all of others policies are combined by combining algorithm. Each combining algorithm returns a single decision either permit or deny and never return both decisions in the same time.

Even a conflict never occurs, it may be interesting to analyse conflict. Here is a scenario. Suppose that a new company is established as a merge from two different companies and as a consequence, all access control policies for the new company come from merging all policies from different companies. Suppose we have following policies.

P1: An employee is allowed to access a company's document only during working hours (8:00 - 17:00), otherwise, the access is denied.

P2: An employee is allowed to access a company's document as long as he or she in the office, otherwise, the access is denied.

Suppose there is a request from an employee at 7:00 from his office. Here we see a conflicted decision, i.e., the access is denied based on policy $P1$ and based on policy $P2$ the access is granted. The final decision depends on the combining algorithm used. If the policy administrator used permit-overrides, then the access is granted, the same case follows for deny-overrides. However, if the only-one-applicable is used, then, the final decision is undecidable (indeterminate).

The purpose of this section is to show an analysis tool to detect possible conflicted policies in XACML policies. We show how to detect conflicted Rules in Section 8.5.1. The same method can be applied for Policy and PolicySet elements.

In addition, in Section 8.5.2 we propose an extension of XACML semantics that can handle conflicts in case OASIS finds it useful to evolve XACML in this direction.

8.5.1 Analysing Conflict Between Rules

We consider conflicted policies from the smallest entity, i.e., Rule element. Here, we formally define a conflict is as follows:

$$\textbf{conflict}: \exists Q : [\![R]\!](Q) = \texttt{permit} \wedge [\![R']\!](Q) = \texttt{deny} \tag{8.4}$$

In order to compute whether there is a conflict in the set of policies, we encode a logic program for conflict property as follows:

$$\begin{array}{ll} \mathcal{P}_{conflict}: & \\ conflict & \leftarrow val(R, \texttt{permit}), val(R', \texttt{deny}), R \neq R'. \\ \bot & \leftarrow \text{not } conflict. \end{array}$$

The same as gap condition, we force ASP solver to find a conflict by putting a constraint $\bot \leftarrow \text{not } conflict$.

A conflict can be analysis whenever $\mathcal{P} = \mathcal{P}_{XACML} \cup \mathcal{P}_{generate\_one} \cup \mathcal{P}_{conflict}$ returns answer sets. The returning models are evidences where the conflict between Rule occurs. We conclude that a set of policies is conflict-free if and only if program $\mathcal{P}$ is unsatisfied, i.e., there is no returned model.

8.5.2 Introducing Conflict In XACML

To this end we add the three pairwise policy values to $\mathbf{P}$ that were briefly mentioned in Subsection 5.3.1: deny with indeterminate permit ($[1, \frac{1}{2}]$), permit with indeterminate deny ($[\frac{1}{2}, 1]$) and conflict ($[1, 1]$) and we write $\mathbf{P}_9 = \mathbf{P} \cup \{ [1, \frac{1}{2}], [\frac{1}{2}, 1], [1, 1] \}$ for the *extended pairwise policy values* obtained in this way. The ordering defined on $\mathbf{P}_9$ is shown in Figure 8.2 and it is easy to see that $\mathbf{P}_9$ forms a lattice (which we may also denote $\mathcal{L}_9$) where the top element is $[1, 1]$ and the bottom element is $[0, 0]$. The ordering of this lattice is the same as $\sqsubseteq_{\mathbf{P}}$ where the least upper bound and the greatest lower bound for $S \subseteq \mathbf{P}_9$ are defined as follows:

$$\bigsqcup_{\mathcal{L}_9} S = Max_{\sqsubseteq_{\mathbf{P}}}(S) \qquad \bigsqcap_{\mathcal{L}_9} S = Min_{\sqsubseteq_{\mathbf{P}}}(S)$$

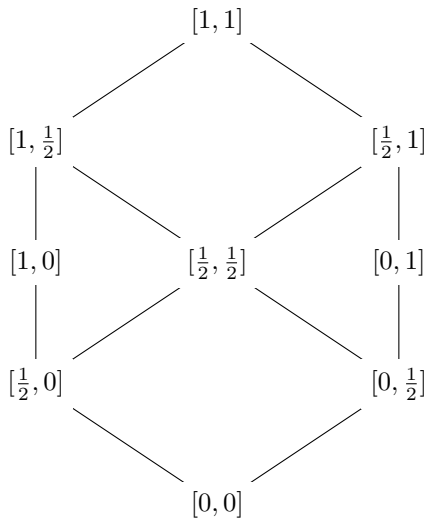

Figure 8.2: The partially ordered set $\mathbf{P}_9$ for extended pairwise policy values.

In this way we capture the conflicts that can be dealt with using the approaches of Belnap 4-valued logic [BDH07, BH08] and $\mathcal{D}$-Algebra 8-valued logic [NBL09] but retain the advantage that our formalization of the conflict-free fragment is consistent with the current XACML 3.0 standard.

8.6 Analysis on Relevant Policies

A policy is relevant if there is a request such that the decision is made based on this policy. Usually in a big set of policies, there is a policy that is irrelevant. This happens because policies are built based on several components and combined together. We formally define a relevant property as follows:

$$\textbf{relevant}(\mathcal{R})\text{: } \exists \mathcal{Q} : [\![\mathcal{R}]\!](\mathcal{Q}) \neq \mathsf{na} \tag{8.5}$$

where Q is Request element and $\mathcal{R}$ is Rule element.

The encoding of relevant property in logic program is as follows:

$$\begin{array}{l} \mathcal{P}_{relevant} : \\ relevant(R) \quad \leftarrow val(R, E), E \neq \mathsf{na}. \end{array}$$

Logically, we can say a Rule is irrelevant as the opposite of 8.5. Formally, we

say that a Rule is irrelevant

$$\textbf{irrelevant(R)}: \forall \mathcal{Q} : [\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{na} \tag{8.6}$$

where Q is Request element and $\mathcal{R}$ is Rule element.

The encoding is as follows.

$$\begin{array}{ll} \mathcal{P}_{irrelevant}: & \\ irrelevant(R) & \leftarrow \text{not } relevant(R). \end{array}$$

Suppose we have following policies.

R1: All programmers can assess source code during office hours, otherwise, the access is denied.

R2: All programmers can access source code 24 hours.

Either $R1$ or $R2$ could be irrelevant depends on the combining algorithm that is used to combine them.

- Permit-overrides and legacy-permit-overrides. We can say that $R1$ is irrelevant since for every Request whenever $R1$ is applicable, $R2$ is applicable. Moreover, for the Request outside office hours, $R1$ gives deny and $R2$ gives permit. However, since the permit-overrides combining algorithm is used, thus, the final decision is permit which comes from $R2$. Hence, we are safe to remove $R1$.
- Deny-unless-permit. The same reason as permit-overrides, we can say that $R1$ is irrelevant.
- Deny-overrides and legacy-deny-overrides. We can say this time that $R2$ is irrelevant. The same reason as above. The difference is that since we use the deny-overrides combining algorithm, thus, the deny takes precedence. For the case that an access request outside office hours, the final decision comes from $R1$ which gives deny decision.
- Permit-unless-deny. The same reason as deny-overrides, we can say that $R2$ is irrelevant.
- First-applicable. We can say that $R2$ is irrelevant since the first policy which is applicable is always $R1$.

Here we extend our definition of irrelevant policies as follows.

Definition 8.5 (irrelevant Policy) We define that a policy is *irrelevant* if for every Request Q:

1. It always return na.

$$\textbf{irrelevant(R)}\text{: } \forall Q : [\![R]\!](Q) = \mathsf{na}$$

2. In case of permit-overrides combining algorithm and deny-unless-permit combining algorithm, a policy is irrelevant if its decision is deny but the final decision of the root policy is permit.

$$\textbf{irrelevant(R)}\text{: } \forall Q : ([\![R]\!](Q) = \mathsf{d}) \wedge ([\![P]\!](Q) = \mathsf{p})$$

where $P = [\mathtt{po}; T; \langle \ldots, R, \ldots \rangle]$ or $P = [\mathtt{dup}; T; \langle \ldots, R, \ldots \rangle]$

3. In case of deny-overrides combining algorithm and permit-unless-deny combining algorithm, a policy is irrelevant if its decision is permit but the final decision of the root policy is deny.

$$\textbf{irrelevant(R)}\text{: } \forall Q : ([\![R]\!](Q) = \mathsf{p}) \wedge ([\![P]\!](Q) = \mathsf{d})$$

where $P = [\mathtt{do}; T; \langle \ldots, R, \ldots \rangle]$ or $P = [\mathtt{pud}; T; \langle \ldots, R, \ldots \rangle]$

4. In case of only-one-applicable combining algorithm, a policy is irrelevant if it is applicable but the final decision of the root policy is Indeterminate. This indicates that there is another policy that is also applicable.

$$\textbf{irrelevant(R)}\text{: } \forall Q : ([\![R]\!](Q) \neq \mathsf{na}) \wedge ([\![P]\!](Q) = \mathsf{idt})$$

where $P = [\mathtt{ooa}; T; \langle \ldots, R, \ldots \rangle]$

5. In case of first-applicable combining algorithm, a policy is irrelevant if it is applicable but there is another policy in the same collection that is in the earlier of the sequence that is also applicable.

$$\textbf{irrelevant}(R_j)\text{: } \forall Q : [\![R_j]\!] \neq \mathsf{na} \wedge [\![R_i]\!](Q) \neq \mathsf{na} \wedge i < j$$

where $P = [\mathtt{fa}; T; \langle \ldots, R_i, \ldots, R_j, \ldots \rangle]$

Deleting Irrelevant Policies. We show that deleting irrelevant policies does not give different result from original PolicySet.

Definition 8.6 (Policy Equivalent) Let $P1$ and $P2$ be XACML Policies. We define that $P1$ is *equivalent* with $P2$ (denoted by $P1 == P2$) as follows:

$$P1 == P2 \Leftrightarrow \forall Q : [\![P1]\!](Q) = [\![P2]\!](Q)$$

Definition 8.7 (Adding A New Rule) Let $P = [\text{comb}_{id}; T; \langle R_1, \ldots, R_n \rangle]$ be an XACML Policy and R be an XACML Rule. We define that P *adds* R in index i (denoted by $P \cup^i R$) as follows:

$$P \cup^i R = [\text{comb}_{id}; T; \langle R_1, \ldots, R_{i-1}, R, R_i, \ldots, R_n \rangle] \quad 1 \leq i \leq (n+1)$$

Definition 8.8 (Strong Policy Equivalent) Let $P1$ and $P2$ be XACML Policies and R be an XACML Rule. We define that $P1$ is *strong equivalent* with $P2$ (denoted by $P1 ==_S P2$) as follows:

$$P1 ==_S P2 \Leftrightarrow \forall R : [\![P1 \cup^* R]\!](Q) == [\![P2 \cup^* R]\!](Q)$$

Proposition 8.9 *Let P be an XACML Policy. If $\mathbb{R}$ be a set of irrelevant Rules, then*

$$P ==_S P \backslash \mathbb{R}$$

where $\backslash$ is a notation to remove all Rules in $\mathbb{R}$

Discussion on the implementation. Kolovski et al. [KH07] presented an analysis on policy redundancy which similar to the irrelevant policies. One way to implement this analysis is by performing change impact analysis for each policy. Pick one policy and check whether removing the policy does or does not give impact to the whole policy set. This way is too verbose. However they also presented a more optimal way how to check impact analysis. This method is in imperative way which is different from our approach in declarative way. A proper solution of implementing this analysis would be our future work.

Chapter 9

Access Control Policies in Smart Grid from Smart Meter Perspective

> To invent, you need a good imagination and a pile of junk.
>
> Thomas A. Edison

Denmark has a long tradition for the promotion of energy efficiency improvement. In future, energy must be used more efficiently and effectively as well as more intelligently so that the increase share of wind energy and emerging growth in solar energy can be used to the widest possible extend to cover the new energy consumption. In 2020, wind power will cover half of Danish electricity consumption and it is expected that a relatively large percentage of overall Danish energy consumption, including for transport and heating, will be electricity-based up to 2020. Therefore, the parties to the agreement decided to draw up a strategy for the smart electricity grid. In autumn 2010 a Smart Grid Network was set up with a number of important players who were to make recommendation for how the electricity sector and the authorities could promote Smart Grid development.

Firstly we introduce a glance of Smart Grid and its components in Section 9.1 and Section 9.2, respectively. In Section 9.3 describe communication in a Smart Grid. Then, in Section 9.4 we present an architecture for access control in Smart Grid from the point of view of Smart Meter – a electric measurement device installed at an energy consumer's house or facility. Finally we present examples of access control policies in Section 9.5.

9.1 Smart Grid in A Nutshell

Electricity is the fundamental and the *crème-de-la-crème* (the best) of energy resource in our economy and social life. Without electricity, life will be uncomfortable, our industry will find difficulty and the production will be slow.

The history of commercialisation of electric power began in the early 20th century. The demand of electric power rocketed due to light bulb revolution and the promise of the electric motor. At first, small utility companies provided power to local industrial plants and private communities. Some larger businesses even generated their own power. Seeking greater efficiency and distribution, utility companies pooled their resources, sharing transmission lines and quickly forming electrical networks called grids.

A *Smart Grid* is the electricity grid of the future: it is expected to be more reliable, more economical, more efficient, and more secure than the standard grid of today. It uses communication technology and information technology (IT) to link all components to the power grid, including generating stations, distribution facilities, transformers, businesses and households. It is a system intended to enable the stable supply and efficient usage of electrical power. The latest technology is used to give "intelligent" functions to the entire power distribution grid, making it a "smart" grid capable of reducing emissions of greenhouse gases and boosting energy efficiency.

Due to its inherent importance for society, its dimension, and to the fact that massive deployment is expected in the near future, the Smart Grid is the most popular example of Cyber-Physical System (CPS). Such systems aim at monitoring physical processes by means of an interconnected network of sensors. Measurements made by these sensors are used to act on the sensed environment in order to optimise an operational goal. The physical layer of the Smart Grid consists of the existing grid infrastructure that is enhanced with intelligent devices (e.g. smart meters) which are able to exchange messages across a communication layer.

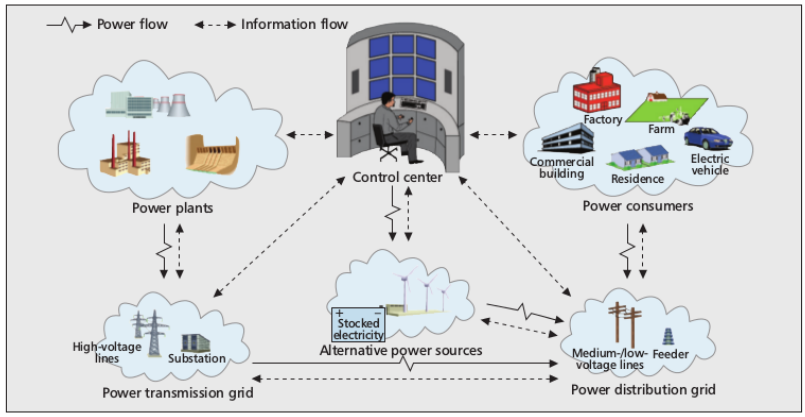

Figure 9.1: A Smart Grid Schema in A Nutshell [LLL+12].

One of the most interesting security-related characteristics of CPS such as the Smart Grid, is in the clear distinction between the physical layer and the cyber layer. There are two control flows in a Smart Grid, i.e., power flow (or electric flow) and information flow (or data flow) [NIS12]. Figure 9.1 illustrates control flows in Smart Grid.

Power Flow. Energy providers such as power plants and wind turbines generates electric power. Later, the power is transported via the transmission network (high-voltage) into a distribution grid. The Distribution Service Operator (DSO) transports power from the transmission network to its customers (businesses and households). In Smart Grid, each customer has Smart Meter (SM) installed on site which can (dis-) connect power supply or load limitation. When it is low-season, SM can send electricity from Solar Panel, or from the grid to Rechargeable Battery (RB) in order to save some electricity. When the RB is full, the SM can either use the electricity for fulfilling energy usage at home or release the power to the grid.

Information Flow. The information flow always occurs both direction between parties. In particular in areas with high risk of overloading, the grid companies may utilise the existing grid more effectively and closer to the fullest of its capacity by installing SM, which will allow them to receive real-time data about the status of and load on the grid. In combination with efforts to move consumption to off-peak hours, this may prevent or postpone the need for further grid investments. There are two kinds of data i.e., *aggregated data* – data is presented in a summary form, and *detailed data* – data contains detail information regarding time, kind of appliance, who has used, etc. Only the owner of SM,

i.e., the customer where the SM is installed who can access the detailed data due to privacy reason. The SM only send data aggregation to DSO.

Converting the electricity grid into a CPS enables more efficient and reliable grid operations, and thus leads to a great many benefits, ranging from energy savings to a high degree of home automation. Despite the many benefits of the Smart Grid, it exposes the system to many new threats. First, numerous malicious parties have an interest in compromising such a system, due to the value of the information flowing in the cyber layer and the impact on physical components. Secondly, due to the complexity of the system, also the number of elements that need protection increases: in the standard grid only the physical layer is subject to attack, while in the Smart Grid also the cyber layer is under threat. What is more, the two layers constantly interact, therefore an attack on one level may in fact grant to the adversary control over the whole system. One way to protect the system is by controlling the access. We believe access control can be used as a first step of security to protect the whole smart grid system.

Access control is a simple technique to prevent unauthorised use by mediating how an access to the resource is granted or denied. Controlling who is on the grid is a vital element to the overall security strategy. There are many user groups that are interested to be on the network such as employees, contractors, guests, and even customers. The access should only be granted based on related assets. For example, a customer who has Internet-enabled access is allowed to view energy consumption and bills online.

9.2 Smart Grid Components

According to [NGE11], a conceptual scheme of the Smart Grid consists of the following related components.

The *Distribution Service Operator (DSO)* is in charge of managing the power distribution within the Smart Grid, regulating power consumption through Smart Meters; for instance, the DSO can request a Smart Meter to reduce the overall power consumption in peak hours.

Consumers (C) are the final users of the electricity provided by the DSO, with which a consumer has a supply agreement. A consumer enters into a contract with the DSO simply in order to secure power for his *Electric Household Appliances (EHAs)*, i.e., electrical devices for specific household functions, such as cooking or cleaning. An EHA can be a smart device (SEHA), or a legacy device

(LEHA): the former can be programmed by a consumer or a Smart Meter, and it informs them about the energy usage, while the latter can simply be turned on or off.

Smart Meters (SM) are the physical point-of-contact between the DSO and a consumer: the energy provided by the DSO to a consumer in order to operate EHAs is physically managed by a Smart Meter located on the consumer's premises. For this reason, each EHA is connected to the SM, which can terminate or re-connect power supply to any EHA remotely, in order to limit electricity consumption. Moreover, a SM can be programmed to maintain a schedule for operation of EHAs, and thus has two sources for input: the DSO, which are global, and the consumer requirements, which are local. The requirements set by the consumer can be violated in order to obey the instructions issued by the DSO; these can be interpreted as hard constraints. It is worth noting, however, that the DSO always issues global instructions, and the SM is in charge of finding the best strategy to comply with this instruction through an intelligent schedule. Finally, similar to a regular meter, it collects information about consumption and forwards it to the DSO.

A consumer may enter into a contract with *Service Providers (SP)* for the maintenance of his EHAs. For instance, a mechanic could provide a remote service for repairing electric vehicles, or a weather forecasting service could compute the expected irradiation of a solar panel, and this information could be exploited to estimate future energy requirements. The communication between a SP and the related EHA(s) relies on the SM, which is in charge of receiving messages from the former and subsequently relaying them to the latter. A Service Provider thus needs to register the EHA and obtain digital certificates for the communication. The validity of such certificates is checked by the Smart Meter when it is asked to contact an EHA on behalf of a Service Provider.

9.3 Communication in Smart Grid

From a high level perspective, the distinguishing trait of the Smart Grid when compared to the regular power grid, is in the information flow that aims at optimising the operational goals of the system. The communication can be partitioned into intra-HAN and extra-HAN links, and the Smart Meter is the gateway that connects these separate worlds. In particular, different technologies can be leveraged to implement the connections between the system's various entities [BBA07, PKS10].

9.3.1 Intra-HAN

Intra-HAN exchange involves communication between the EHAs (they might be connected to each other), and between a single EHA and the Smart Meter.Examples of communication links at this level are the following.

SM-EHA: according to the instructions received from the Electric Utility or the HAN administrator, a Smart Meter can decide to start, shut down, or limit the use of any EHA. The restrictions that can be enforced other than putting the device in 'on' or 'off' state depend on the capabilities and the interface offered by the EHA: a smart washing machine, for example, could offer the Smart Meter the possibility to set the temperature of the water for a washing cycle, with a consequent impact on the energy consumption.

EHA-EHA: in some applications various EHAs need to cooperate in order to provide a service or in order to optimise their efficiency. For example, dish washer is turned on after the washing machine is done its job. This communication exchange can be channeled through the Smart Meter, or the involved EHAs communicate directly with another one.

C-SM/EHA: the consumer is also part of this intra-HAN scenario, as he can act directly on the Smart Meter or on the appliance in question. A consumer can pass a list of activities that need to be accomplished to the Smart Meter, which then computes a schedule so that all of the activities are fulfilled in the optimal way with respect to the consumer's preferences (e.g. minimal total energy cost, percentage of green energy used, less noisy level during the night). In this case, the Smart Meter acts as an intelligent device and not only as a metering device.

9.3.2 Extra-HAN

Extra-HAN exchange involves communication via a Smart Meter between a HAN and the external world, consisting in Service Providers and the Electric Utility. Various Wide Area Network (WAN) technologies such as PLC, Broadband Power Line, GPRS, IEEE 802.16 WiMaX can be used here. PLC is mentioned as the most efficient data transmission scheme on electric power transmission in [DWD11], and this is also the approach taken by the dominant Italian electric utility (Enel SpA), which deployed more than 30 million Smart Meters in the early 2000s. Finally, in [YZN+12] a real case is studied in which

the communication along the high-voltage lines of the grid relies on a Wireless Sensor Network, proving that electromagnetic interference can be dealt with. At this level we notice the following links.

SP-SM-EHA: a Smart Meter registers and pairs a Service Provider with the correct EHAs, thus establishing a bidirectional communication path between a Service Provider and an EHA.

DSO-SM: the Distribution Service Operator sends consumption related instructions to a Smart Meter, and collects consumption reports and other notifications from it. The periodicity of this exchange can vary from 500ms to some minutes. It is important to note that a Smart Meter should only convey aggregated information to the external world, for the sake of privacy.

9.4 Access Control Architecture

There are two main parties that interact in SM, i.e., SM and DSO. The interaction between parties is depicted in Figure 9.2.

Service Interface. The Service Interface functions as a Policy Enforcement Point (PEP). Each party has its own PEP, PEP for SM side and PEP for DSO side. There should be a mechanism that both PEPs can communicate thus, i.e., send access requests and receive decision answers. Each PEP has a connection to their Attribute Service and Policy Authority.

Each PEP can receive an access request using difference languages. For example, the human user can send an access request through browser application or rich client application such as using tablet or smart phone as an interface for PEP in SM side. When a PEP receive an access requests, it transforms the request into XACML form and sends it to Access Control Service in order to get an authorisation answer.

Access Control Service. The Access Control Service functions as Policy Decision Point. It decides whether an access request should be granted or denied based on the access control policies that it has.

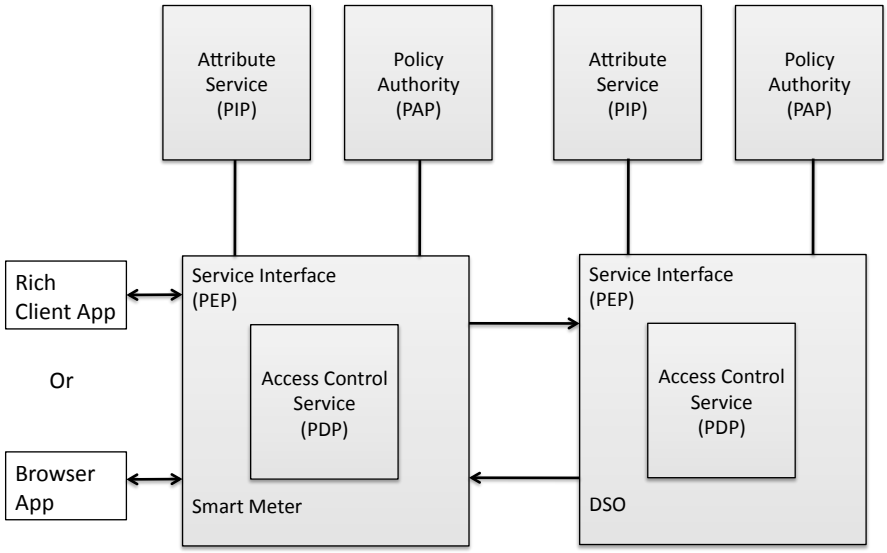

Figure 9.2: Access Control Architecture

Attribute Service. The Attribute Service functions as a Policy Information Point (PIP). The Attribute Service has access to attribute information such as location, purpose of use, object preferences, consent directives and other privacy conditions (object masking, object filtering, user, role, purpose, etc.) that constrain enforcement.

Policy Authority The Policy Authority function as a Policy Administration Point (PAP). The Policy Authority has access to security policies that include rules regarding authorisations required to access a protected resource and additional security conditions (location, time of day, cardinality, separation of duty purpose, etc.) that constrain enforcement.

9.5 Access Control Policies Examples

In this section, we describe a scenario of access control policies in a Smart Grid from Smart Meter perspective. As depicted in Figure 9.3, every object has a connection to the Smart Meter as the central point of view.

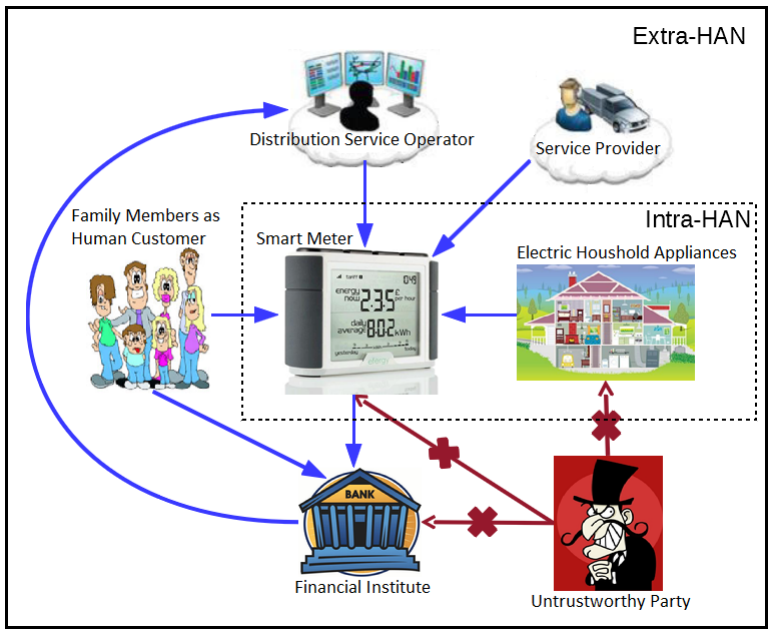

Figure 9.3: Access Control in Smart Grid from Smart Meter Perspective

Scenario from Smart Meter (SM) In this scenario, a Smart Meter can request connection (or disconnection) to (or from) power line from a Distribution Service Operation (DSO). Moreover, the SM has an access to the data (detailed consumption and aggregated consumption) and can update the data periodically. The SM also has an access to the price list from the DSO in order to make a plan how the usage of EHAs. The SM can connect or disconnect EHAs and it is mandatory to write a log about the connection or disconnection time.

Scenario from Human Customer (HC) In this scenario, a Human Customer (HC) can can access detailed data consumption and price list which are stored in the SM. HC can pay the billing statement directly to the Financial Institute (FI). The SM accommodates HC to plan the usage of EHAs.

Scenario from Electric Household Appliances (EHAs) Each Electric Household Appliance (EHA) can connect to the electricity through the SM and it is mandatory that the SM writes a log when the EHA is started. Next, when the EHA stops using the electricity, the SM should write a log again the

disconnection time.

Scenario from Distribution Service Operator (DSO) In this scenario, a Distribution Service Operator (DSO) can see the payment detail from FI. In order to keep the customers' privacy, the DSO is only able to read aggregated data consumption from the SM. The DSO can connect SM to the power line and also can disconnect SM from the power line if the customer has not paid the bill or if there is a technical safety reason.

Scenario from Service Provider (SP) A Service Provider (SP) can read any information of a particular EHA related to its service. The SP can perform a specific action to a particular EHA related to its service.

The first step in developing an access control system is to identify the *objects* that needs protection, the *subjects* which initiate the access requests to the objects and execute activities and the *actions* that can be executed on the objects and must be controlled [SD01]. Thus, we organise the policies based on the resources (objects) that need protection (see Table 9.1).

Table 9.1: Access Control Policies: Objects, Subjects and Actions

Objects	Electric Household Appliance (EHA), Data (Aggregated Data, Detailed Data, and Particular Data), Power Line, Billing Statement, Price List
Subjects	Smart Meter (SM), Human Customer (HC), Distribution Service Operator (DSO), Service Provider (SP), EHA
Actions	read, write, turn on, turn off, connect, disconnect, pay, send, update

9.5.1 Access Control Policies for EHA

1. SM can *turn on* an EHA and it is mandatory to write a log when the EHA is started.

2. SM can *turn off* an EHA and it is mandatory to write a log when the EHA is ended.

3. Through SM, the SP can do a particular action to the corresponding EHA, for example, *update* a new software, and it is mandatory to write the description on the action in the log.

We assume that there is only one request that is allowed each time. However, there might be possibility two policies are applicable, for example, SM received an request from SP in order to turn off a particular EHA. Thus, policy 1 and policy 3 are both applicable. Hence, in this policy set, we prefer to set "first-applicable" combining operator.

9.5.2 Access Control Policies for Data

1. SM can *write* the aggregated data consumption and technical data.
2. DSO can *read* the aggregated data consumption from SM.
3. HC can *read* the detailed consumption data from SM.
4. SP can *read* the data about a particular EHA.

In order to protect data secrecy, we prefer to deny all access request unless we are sure that the requestor has access to it. Hence, we use "deny-unless-permit" combining algorithm.

9.5.3 Access Control Policies for Power Line

1. SM can *connect* to the power line.
2. SM can *disconnect* from the power line.
3. DSO can *connect* SM to the power line.
4. DSO can *disconnect* SM from the power line if the customer has not paid the bill or there is a technical safety reason.
5. EHA can *connect* the electricity through SM and it is mandatory to write a log when the EHA is started.
6. EHA can *disconnect* the electricity through SM and it is mandatory to write a log when the EHA is ended.

In order to make sure that the supply power is always available, any permitted access to the power system should be allowed. In this case, we use "permit-unless-deny" combining operator.

9.5.4 Access Control Policies for Billing Statement

1. HC can *read* the billing statement from the FI.
2. HC can *pay* the billing to the FI.
3. DSO can *send* bill to the FI.
4. DSO can *read* the payment bill from FI.
5. FI can *update* the payment status after the payment transaction succeed

We would like that the bill is always paid and the access to the billing statement is always granted. However, since secrecy is important, we only give permission to the trustworthy requestor. Thus, we use "permit-overrides" combining operator.

9.5.5 Access Control Policies for Price List

1. DSO can *update* the price list to the SM.
2. HC through SM can *see* the price list.
3. SM can *access* the price list in order to make a plan.

The price list is available to public and not secret. Thus, we use "permit-unless-deny" combining operator.

Part V

Conclusion

Chapter 10

Conclusion and Future Work

> Education never ends, Watson. It is a series of lessons, with the greatest for the last.
>
> Sherlock Holmes, His Last Bow

This final chapter contains a summary of our main contributions, viewing them from a broader perspective, and discusses interesting lines of future research.

In the previous parts of the thesis we presented a number of results that bring insights regarding modelling and analysing access control policies in XACML 3.0. Part II introduces an abstract syntax and formal semantics of XACML 3.0. Part III presents a systematic technique for transforming XACML 3.0 policies in Answer Set Programming (ASP). We show that the resulting logic program has a unique answer set that directly corresponds to our formalisation of the standard semantics of XACML 3.0 presented in Part II. Part IV demonstrates how the results from Part III make it possible to use off-the-shelf ASP solvers to formally verify properties of access control policies represented in XACML.

In the following sections we take a closer look at the conclusions drawn from our results. Then we point at desirable future developments.

10.1 Conclusion

Abstraction XACML 3.0. We have introduced XACML 3.0 and distilled the core ingredients of the XACML policy components in the form of an abstract syntax much less verbose than the standard XML based notation. This has allowed us to develop a six valued logic for describing the semantics of the overall policy evaluation as well as the semantics of the eight combining operators that are key components of XACML. Such formalisation is always tricky – the usual imprecision of standards apply to formalisation attempts as well and there always is the risk of incorporating modelling artefacts into the formalisation. We have addressed this challenge by providing two different types of formalisation (one based on V_6 and one based on pairwise policy values $\mathbf{P}$) and shown them to be equivalent (see Figure 5.1). To guard against modelling artefacts we provide an alternative lattice based way of characterising the policy combining operators and we formally prove the equivalence of these approaches thereby increasing our faith in either one.

XACML 3.0 Transformation. We propose a logic-based XACML analysis framework using Answer Set Programming (ASP). With ASP we model an XACML Policy Decision Point (PDP) that loads XACML policies and evaluates XACML requests against these policies. We have modelled the XACML PDP in a declarative way using the ASP technique by transforming XACML 3.0 elements into logic programs. Our transformation of XACML 3.0 elements is directly based on XACML 3.0 semantics and we have shown that the answer set of each program transformation is unique and that it agrees with the semantics of XACML 3.0.

Application. We show how to use ASP for analysing access control security properties. We can help policy developers analyse their access control policies such as checking policies' completeness and verifying policy properties by inspecting the answer set of $\Pi_{\text{XACML}} \cup \Pi_{generator} \cup \Pi_{configuration}$ – the program obtained by transforming XACML 3.0 elements into logic programs joined with a query generator program and a configuration program.

We have discussed how to extend XACML with considerations of conflict. This is a notion that is absent from XACML 3.0 but is present in both of the earlier formalisation mentioned above. This paves the way for extending XACML in a sound way should OASIS decide to incorporate such features. We believe that some of the policy combining operators that one might want to incorporate into an XACML setting, in particular Consensus-Based-Vote, would benefit from such a choice.

We present access control security policies in a Smart Grid from Smart Meter perspective.

10.2 Future Work

As mentioned earlier, there are several aspects of XACML components that we do not consider in this thesis (see Chapter 4). We do not consider Obligations and Advices since they are not mandatory elements in XACML. However, Obligations and Advices are used to enrich the authorisation flow. Studying their behaviour and adding them into our work would bring the modelling of XACML 3.0 more complete.

There is still a lot of room for improvement in the application of ASP to model XACML 3.0. ASP has been successfully applied to solve combinatorial optimisation problems. However, ASP is not directly suitable for modelling problems with continuous domains. In the implementation of XACML 3.0, such problems may occur in may access control policies. For example, the access to an object depends on the time when the request arrived. There are approaches to handle continuous domains such as [VNDCV06, MGZ08, LM13].

The implementation of case study is far from complete. We would like to have more real case studies which we have real access control policies scenarios that we can model.

Appendix A

Proofs: Semantics of XACML Combining Algorithms

> It is at this point that normal language gives up, and goes and has a drink.
>
> Terry Pratchet, The Color of Magic

In the following we present proofs of results from Chapter 5, in particular Proposition 5.8 and Proposition 5.24.

A.1 Proof: Proposition 5.8

Proposition 5.8 *Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in V_6. Then*

$$\delta(\bigoplus_{\text{po}}^{V_6}(\langle s_1, \ldots, s_n \rangle)) = \bigoplus_{\text{po}}^{\mathbf{P}}(\delta(\langle s_1, \ldots, s_n \rangle))$$

PROOF. Let $S = \langle s_1, \ldots, s_n \rangle$ and $S' = \{ s_1, \ldots, s_n \}$.
There are six possible outcomes for $\delta(\bigoplus_{\mathsf{po}}^{V_6}(S)) = \bigoplus_{\mathsf{po}}^{\mathbf{P}}(\delta(S))$:

1. $\delta(\bigoplus_{\mathsf{po}}^{V_6}(S)) = [1,0]$ iff $\bigoplus_{\mathsf{po}}^{V_6}(S) = \top_{\mathsf{d}} = \bigsqcup_{\mathsf{po}} S'$ (by definition of $\bigoplus_{\mathsf{po}}^{V_6}$).
   Based on $\sqsubseteq_{\mathsf{po}}$ we get that $\exists i : s_i = \top_{\mathsf{d}}$ and $\forall j : i \neq j \Rightarrow s_j \in \{ \top_{\mathsf{d}}, I_{\mathsf{d}}, \bot \}$.
   Thus, using the definition of δ we get that $\delta(s_i) = [1,0]$ and $\forall j : i \neq j \rightarrow \delta(s_j) \in \{ [1,0], [\frac{1}{2},0], [0,0] \}$.
   Furthermore we get that $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \}) = [1,0]$.
   Hence, by the definition of $\bigoplus_{\mathsf{po}}^{\mathbf{P}}$ we get that $\bigoplus_{\mathsf{po}}^{\mathbf{P}}(\delta(S)) = [1,0]$.

2. $\delta(\bigoplus_{\mathsf{po}}^{V_6}(S)) = [0,1]$ iff $\bigoplus_{\mathsf{po}}^{V_6}(S) = \top_{\mathsf{p}} = \bigsqcup_{\mathsf{po}} S'$ (by definition of $\bigoplus_{\mathsf{po}}^{V_6}$).
   Based on $\sqsubseteq_{\mathsf{po}}$ we get that $\exists i : s_i = \top_{\mathsf{p}}$.
   Thus, by the definition of δ we get that $\delta(s_i) = [0,1]$.
   Furthermore we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \} = [\_, 1])$.
   Hence, by the definition of $\bigoplus_{\mathsf{po}}^{\mathbf{P}}$ we get $\bigoplus_{\mathsf{po}}^{\mathbf{P}}(\delta(S)) = [0,1]$.

3. $\delta(\bigoplus_{\mathsf{po}}^{V_6}(S)) = [\frac{1}{2},\frac{1}{2}]$ iff $\bigoplus_{\mathsf{po}}^{V_6}(S) = I_{\mathsf{dp}} = \bigsqcup_{\mathsf{po}} S'$ (by definition of $\bigoplus_{\mathsf{po}}^{V_6}$).
   Based on $\sqsubseteq_{\mathsf{po}}$ there are three cases:

   (a) $\exists i : s_i = I_{\mathsf{dp}}$ and $\forall j : j \neq i \Rightarrow s_j \in \{ I_{\mathsf{dp}}, I_{\mathsf{p}}, \top_{\mathsf{d}}, I_{\mathsf{d}}, \bot \}$.
   Hence, by the definition of δ we get that $\delta(s_i) = [\frac{1}{2},\frac{1}{2}]$ and $\forall s_j : \delta(s_j) \in \{ [\frac{1}{2},\frac{1}{2}], [0,\frac{1}{2}], [1,0], [\frac{1}{2},0], [0,0] \}$.
   Furthermore we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \}) = [D, \frac{1}{2}]$ where $D \geq \frac{1}{2}$.
   Therefore, by the definition of $\bigoplus_{\mathsf{po}}^{\mathbf{P}}$ we get that $\bigoplus_{\mathsf{po}}^{\mathbf{P}}(\delta(S)) = [\frac{1}{2},\frac{1}{2}]$.

   (b) $\exists i,j : s_i = I_{\mathsf{p}}, s_j \in \top_{\mathsf{d}}$ and $\forall k : k \neq i, k \neq j, s_k \in \{ I_{\mathsf{p}}, \top_{\mathsf{d}}, I_{\mathsf{d}}, \bot \}$.
   Hence, by the definition of δ we get that $\delta(s_i) = [0,\frac{1}{2}]$ and $\delta(s_j) = [1,0]$ and $\forall k : \delta(s_k) \in \{ [0,\frac{1}{2}], [1,0], [\frac{1}{2},0], [0,0] \}$.
   Therefore, we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \} = [D, \frac{1}{2}])$ where $D \geq \frac{1}{2}$.
   Moreover, by the definition of $\bigoplus_{\mathsf{po}}^{\mathbf{P}}$ we get that $\bigoplus_{\mathsf{po}}^{\mathbf{P}}(\delta(S)) = [\frac{1}{2},\frac{1}{2}]$.

   (c) $\exists i,j : s_i = I_{\mathsf{p}}, s_j \in I_{\mathsf{d}}$ and $\forall k : k \neq i, k \neq j, s_k \in \{ I_{\mathsf{p}}, I_{\mathsf{d}}, \bot \}$.
   Hence, by the definition of δ we get that $\delta(s_i) = [0,\frac{1}{2}]$ and $\delta(s_j) = [1,0]$ and $\forall k : \delta(s_k) \in \{ [0,\frac{1}{2}], [\frac{1}{2},0], [0,0] \}$.
   Hence, we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \} = [D,1])$ where $D \geq \frac{1}{2}$.
   Moreover, by the definition of $\bigoplus_{\mathsf{po}}^{\mathbf{P}}$ we get that $\bigoplus_{\mathsf{po}}^{\mathbf{P}}(\delta(S)) = [\frac{1}{2},\frac{1}{2}]$.

4. $\delta(\bigoplus_{\mathsf{po}}^{V_6}(S)) = [\frac{1}{2},0]$ iff $\bigoplus_{\mathsf{po}}^{V_6}(S) = I_{\mathsf{d}} = \bigsqcup_{\mathsf{po}} S'$ (by definition of $\bigoplus_{\mathsf{po}}^{V_6}$).
   Based on $\sqsubseteq_{\mathsf{po}}$ we get that $\exists i : s_i = I_{\mathsf{d}}$ and $\forall j : j \neq i, s_j \in \{ I_{\mathsf{d}}, \bot \}$.
   Hence, by the definition of δ we get that $\delta(s_i) = [\frac{1}{2},0]$ and $\forall j : \delta(s_j) \in$

$\{ [\frac{1}{2}, 0], [0, 0] \}$.
Furthermore we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \} = [\frac{1}{2}, 0])$.
Therefore, by the definition of $\bigoplus^{\mathbf{P}}_{\mathsf{po}}$ we get that $\bigoplus^{\mathbf{P}}_{\mathsf{po}}(\delta(S)) = [\frac{1}{2}, 0]$.

5. $\delta(\bigoplus^{V_6}_{\mathsf{po}}(S)) = [0, \frac{1}{2}]$ iff $\bigoplus^{V_6}_{\mathsf{po}}(S) = I_{\mathsf{p}} = \bigsqcup_{\mathsf{po}} S'$ (by definition of $\bigoplus^{V_6}_{\mathsf{po}}$).
Based on $\sqsubseteq_{\mathsf{po}}$ we get that $\exists i : s_i = I_{\mathsf{p}}$ and $\forall j : j \neq i, s_j \in \{ I_{\mathsf{p}}, \bot \}$.
Hence, by the definition of δ we get that $\delta(s_i) = [0, \frac{1}{2}]$ and $\forall j : \delta(s_j) \in \{ [0, \frac{1}{2}], [0, 0] \}$.
Furthermore we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \} = [0, \frac{1}{2}])$.
Therefore, by the definition of $\bigoplus^{\mathbf{P}}_{\mathsf{po}}$ we get that $\bigoplus^{\mathbf{P}}_{\mathsf{po}}(\delta(S)) = [0, \frac{1}{2}]$.

6. $\delta(\bigoplus^{V_6}_{\mathsf{po}}(S)) = [0, 0]$ iff $\bigoplus^{V_6}_{\mathsf{po}}(S) = \bot = \bigsqcup_{\mathsf{po}} S'$ (by definition of $\bigoplus^{V_6}_{\mathsf{po}}$).
Based on $\sqsubseteq_{\mathsf{po}}$ we get that $\forall i : s_i = \bot$.
Hence, by the definition of δ we get that $\forall i : \delta(s_i) = [0, 0]$.
Furthermore we get $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \} = [0, 0])$.
Therefore, by the definition of $\bigoplus^{\mathbf{P}}_{\mathsf{po}}$ we get that $\bigoplus^{\mathbf{P}}_{\mathsf{po}}(\delta(S)) = [0, 0]$. □

A.2 Proof: Proposition 5.24

Proposition 5.24 *Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in V_6. Then*

$$\delta(\bigoplus^{V_6}_{\mathsf{ooa}}(\langle s_1, \ldots, s_n \rangle)) = \bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(\langle s_1, \ldots, s_n \rangle))$$

Proof. Let $S = \langle s_1, \ldots, s_n \rangle$.
There are four possible outcomes for $\delta(\bigoplus^{V_6}_{\mathsf{ooa}}(S)) = \bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S))$:

1. $\delta(\bigoplus^{V_6}_{\mathsf{ooa}}(S)) = [1, 0]$ iff $\bigoplus^{V_6}_{\mathsf{ooa}}(S) = \top_{\mathsf{d}}$.
We get that $\exists i : s_i = \top_{\mathsf{d}}$ and $\forall j : i \neq j \Rightarrow s_j = \bot$ based on the definition of $\bigoplus^{V_6}_{\mathsf{ooa}}(S)$.
Thus, $\exists i : \delta(s_i) = [1, 0]$ and $\forall j : i \neq j \Rightarrow \delta(s_j) = [0, 0]$.
Hence, $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \}) = [1, 0]$ and
$\forall j : i \neq j \Rightarrow Min_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_i), \delta(s_j) \}) = [0, 0]$.
Therefore, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [1, 0]$.

2. $\delta(\bigoplus^{V_6}_{\mathsf{ooa}}(S)) = [0, 1]$ iff $\bigoplus^{V_6}_{\mathsf{ooa}}(S) = \top_{\mathsf{p}}$.
We get that $\exists i : s_i = \top_{\mathsf{p}}$ and $\forall j : i \neq j \Rightarrow s_j = \bot$ based on the definition

of $\bigoplus^{V_6}_{\mathsf{ooa}}(S)$.
Thus, $\exists i : \delta(s_i) = [0,1]$ and $\forall j : i \neq j \Rightarrow \delta(s_j) = [0,0]$.
Hence, $Max_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_1), \ldots, \delta(s_n) \,\}) = [0,1]$ and
$\forall j : i \neq j \Rightarrow Min_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_i), \delta(s_j) \,\}) = [0,0]$.
Therefore, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [0,1]$.

3. $\delta(\bigoplus^{V_6}_{\mathsf{ooa}}(S)) = [\frac{1}{2}, \frac{1}{2}]$ iff $\bigoplus^{V_6}_{\mathsf{ooa}}(S) = I_{\mathsf{dp}}$.
Based on the definition of $\bigoplus^{V_6}_{\mathsf{ooa}}(S)$, there are two possibilities:

   (a) $\exists i : s_i \in \{\, I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}} \,\}$. Thus, $\exists i : \delta(s_i) \in \{\, [\frac{1}{2}, 0], [0, \frac{1}{2}], [\frac{1}{2}, \frac{1}{2}] \,\}$.
   Therefore, $Max_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_1), \ldots, \delta(s_n) \,\}) = [D, P], D \geq \frac{1}{2} \vee P \geq \frac{1}{2}$.
   There are four possibilities:

      i. $D = \frac{1}{2}$.
      Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

      ii. $D = 1$.
      Hence, there is $\delta(s_j) = [1,0]$. Moreover,
      - for $\delta(s_i) = [\frac{1}{2}, 0] \vee [\frac{1}{2}, \frac{1}{2}]$, $Min_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_i), \delta(s_j) \,\}) = [1,0]$ and $max(\{\, 1, 0 \,\}) \geq \frac{1}{2}$.
        Thus, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.
      - for $\delta(s_i) = [0, \frac{1}{2}]$, $Max_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_1) \,\}, \ldots, \delta(s_n)) = [D, P], D = 1 \wedge P \geq \frac{1}{2}$.
        There are two possibilities:
        - $P = \frac{1}{2}$.
          Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.
        - $P = 1$.
          Since $D = P = 1$, thus, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

      iii. $P = \frac{1}{2}$.
      Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

      iv. $P = 1$.
      Hence, there is $\delta(s_j) = [0,1]$. Moreover,
      - for $\delta(s_i) = [0, \frac{1}{2}] \vee [\frac{1}{2}, \frac{1}{2}]$, $Min_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_i), \delta(s_j) \,\}) = [0,1]$ and $max(\{\, 1, 0 \,\}) \geq \frac{1}{2}$. Thus, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.
      - for $\delta(s_i) = [\frac{1}{2}, 0]$, $Max_{\sqsubseteq_{\mathbf{P}}}(\{\, \delta(s_1) \,\}, \ldots, \delta(s_n)) = [D, P], D \geq \frac{1}{2} \wedge P = 1$. There are two possibilities:
        - $D = \frac{1}{2}$. Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.
        - $D = 1$. Since $D = P = 1$, thus, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathsf{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

(b) $\exists i, j : i \neq j$ and $s_i, s_j \in \{ \top_{\mathtt{d}}, \top_{\mathtt{p}} \}$.
Thus, $\exists i, j : 1 \neq j$ and $\delta(s_i), \delta(s_j) \in \{ [1,0], [0,1] \}$.
Therefore, there are three possibilities:

i. $\delta(s_i) = [1,0]$ and $\delta(s_j) = [1,0]$.
Moreover, $Min_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_i), \delta(s_j) \}) = [1,0]$ and $max(\{ 1, 0 \}) \geq \frac{1}{2}$.
Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathtt{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

ii. $\delta(s_i) = [1,0]$ and $\delta(s_j) = [0,1]$.
Moreover, $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \}) = [D, P], D = P = 1$.
Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathtt{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

iii. $\delta(s_i) = [0,1]$ and $\delta(s_j) = [0,1]$.
Moreover, $Min_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_i), \delta(s_j) \}) = [0,1]$ and $max(\{ 1, 0 \}) \geq \frac{1}{2}$.
Hence, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathtt{ooa}}(\delta(S)) = [\frac{1}{2}, \frac{1}{2}]$.

4. $\delta(\bigoplus^{V_6}_{\mathtt{ooa}}(S)) = [0,0]$ iff $\bigoplus^{V_6}_{\mathtt{ooa}}(S) = \bot$.
Based on the definition of $\bigoplus^{V_6}_{\mathtt{ooa}}(S)$ we get that $\forall i : s_i = \bot$.
Thus, $\forall i : \delta(s_i) = [0,0]$. Therefore, $Max_{\sqsubseteq_{\mathbf{P}}}(\{ \delta(s_1), \ldots, \delta(s_n) \}) = [0,0]$.
Therefore, based on the definition, $\bigoplus^{\mathbf{P}}_{\mathtt{ooa}}(\delta(S)) = [0,0]$.

Appendix B

Proof of Semantics Correlation Between XACML-ASP And XACML 3.0

> Eliminate all other factors, and the one which remains must be the truth.
>
> Sherlock Holmes, The Sign of Four

In the following we present proofs of results from Chapter 7. Recall that an answer set (AS) M of program Π is a a stable model of Π^M and it is a minimal model. Hence, all atoms in M are *true* with respect to M and all atoms not in M are *false* with respect to M.

Lemma B.1 *Let M be an AS of program Π and let $H \leftarrow Body$ be a clause in Π. Then, $H \in M$ if $M(Body) = \top$.*

Proof. Let $Body = B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n$. To show the lemma holds, suppose $M(Body) = \top$. Then we find that $\{ B_1, \ldots, B_m \} \subseteq M$ and

$M \cap \{ B_{m+1}, \ldots, B_n \} = \emptyset$. Since M is a stable model of Π^M then we find that $H \leftarrow B_1, \ldots, B_n$ is in Π^M. Since M is a model of Π^M and $\{ B_1, \ldots, B_m \} \subseteq M$ then $M(H) = \top$. Thus $H \in M$. □

The Lemma B.1 only ensures that if the body of a clause is true under an AS M then the head is also in M. However, in general, if the head of a clause is in an AS M then there is no guarantee that the body is always true under M. For example, suppose we have a program $\{ p \leftarrow \top., p \leftarrow q. \}$. In this example the only AS is $M = \{ p \}$. We can see that p is in M. However, q is not in M, thus, $M(q)$ is false.

Lemma B.2 *Let M be an AS of program Π and let H be in M. Then, there is a clause in Π where H as the head.*

Proof. Suppose that M is an AS of program Π. Then we find that M is model of Π^M. Suppose $H \in M$ and there is no clause in Π^M such that H as the head. Then, we find that $M' = M / \{ H \}$ and M' is a model of Π^M. Since M is a minimal model of Π^M but we have $M' \subset M$. Therefore we find a contradiction. Thus, there should be a clause in Π^M such that H as the head. Hence, there is a clause in Π such that H as the head. □

Lemma B.3 *Let M be an AS of program Π and let H be in M. Then, there exists a clause where H as the head and the body is true under M.*

Proof. Suppose that M is an AS of program Π and $H \in M$. By Lemma B.2, we find that there is a clause in Π in a form $H \leftarrow Body$. Suppose that $M(Body) \neq \top$. Therefore, $H \leftarrow Body$ is not in Π^M. Moreover, we can find another interpretation M' such that $M / \{ H \}$ and M' is also a model of Π^M. However, we know that M is a minimal model for Π^M but we have $M' \subset M$. Thus, there is a contradiction. □

Lemma B.4 *Let M be an AS of program Π and let H be a head of a clause in Π. Then,*

$$M(H) = \bot \qquad \textit{if and only if} \qquad \forall i : M(Body_i) = \bot$$

where $Body_i$ is a body of a clause where H is the head.

Proof. ($\Rightarrow$) It is straightforward since M is a model of Π.

($\Leftarrow$) Suppose that $\forall i : M(Body_i) = \bot$ and $M(H) = \top$. Therefore, $H \in M$. Moreover, we can find another interpretation M' such that $M / \{ H \}$ and M' is also a model of Π. However, we know that M is a minimal model for Π^M but we have $M' \subset M$. Thus, there is a contradiction. $\square$

We define new notations that we will use for the rest of the proofs.

- Let $\Pi_{\mathcal{Q}}$ be a program obtained from Request $\mathcal{Q}$ transformation.
- Let $\Pi_{\texttt{err}}$ be a program indicating the `error` occurrence. $\Pi_{\texttt{err}} = \emptyset$ if there is no `error` and $\Pi_{\texttt{err}} = \{ \texttt{error} \leftarrow \top. \}$ otherwise.
- Let $\Pi_{\texttt{eval}_M}$ be a program obtained from $\texttt{eval}_M$ transformation.
- Let $\Pi_{\mathcal{M}}$ be a program obtained by transforming Match $\mathcal{M} = f_{MatchID}(v, cat)$ evaluation into a logic program.
- Let $\mathcal{A} = \texttt{allof}(\mathcal{A}_{id}) : \{ \mathcal{M}_1, \ldots, \mathcal{M}_n \}$ be an AllOf component.
  - $\Pi_{\mathcal{A}}$ denotes a logic program obtained by transforming $\mathcal{A}$ evaluation into a logic program.
  - $\Pi^{\mathcal{A}} = \bigcup \Pi_{\mathcal{M}_i} \cup \Pi_{\mathcal{A}}$ denotes a logic program obtained by joining all logic programs obtained by evaluation transformation from each component in $\mathcal{A}$, including the transformation of $\mathcal{A}$ itself.
    Remark: We use capital font to denote interpretations and calligraphic font to denote XACML components. Please note that the symbol Π^A denotes a reduct program with respect to A and $\Pi^{\mathcal{A}}$ denotes as explained above.
- For the rest, see table below.

Match $\mathcal{M}$ $\Pi^{\mathcal{M}} = \Pi_{\texttt{err}} \cup \Pi_{\texttt{eval}_M} \cup \Pi_{\mathcal{M}}$
AllOf $\mathcal{A} = \texttt{allof}(\mathcal{A}_{id}) : \{ \mathcal{M}_1, \ldots, \mathcal{M}_n \}$ $\Pi^{\mathcal{A}} = \bigcup \Pi^{\mathcal{M}_i} \cup \Pi_{\mathcal{A}}$
AnyOf $\mathcal{E} = \texttt{anyof}(\mathcal{E}_{id}) : \{ \mathcal{A}_1, \ldots, \mathcal{A}_n \}$ $\Pi^{\mathcal{E}} = \bigcup \Pi^{\mathcal{A}_i} \cup \Pi_{\mathcal{E}}$
Target $\mathcal{T} = \texttt{target}(\mathcal{T}_{id}) : \{ \mathcal{E}_1, \ldots, \mathcal{E}_n \}$ $\Pi^{\mathcal{T}} = \bigcup \Pi^{\mathcal{E}_i} \cup \Pi_{\mathcal{T}}$
Condition $\mathcal{C}$ $\Pi_{\mathcal{C}}$
Rule $\mathcal{R} = \texttt{rule}(\mathcal{R}_{id}) : \{ E \ ; \ \mathcal{T} \ ; \ \mathcal{C} \}$ $\Pi^{\mathcal{R}} = \Pi^{\mathcal{T}} \cup \Pi^{\mathcal{C}} \cup \Pi_{\mathcal{R}}$
Policy $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{ \bigoplus_{id} \ ; \ \mathcal{T} \ ; \ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \}$ $\Pi^{\mathcal{P}} = \bigcup \Pi^{\mathcal{R}_i} \cup \Pi^{\mathcal{T}} \cup \Pi^{\oplus_{id}} \cup \Pi_{\mathcal{P}}$
PolicySet $\mathcal{PS} = \texttt{policyset}(\mathcal{PS}_{id}) : \{ \bigoplus_{id} \ ; \ \mathcal{T} \ ; \ \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle \}$ $\Pi^{\mathcal{PS}} = \bigcup \Pi^{\mathcal{P}_i} \cup \Pi^{\mathcal{T}} \cup \Pi^{\text{comb}_{id}} \cup \Pi_{\mathcal{PS}}$
Combining Algorithm $\bigoplus_{id}$ is either `do`, `po`, `dup`, `pud`, `fa`, `ooa`, `ldo`, or `lpo` $\Pi^{\oplus_{id}} = \bigcup \Pi_{\mathcal{R}_i} \cup \Pi_{\mathcal{P}_{id}} \cup \Pi_{\bigoplus_{id}}$

The map of the set $\{ \top, \bot, I \}$ into each component's value is as following.

V_3	Match, AllOf, AnyOf, Target	Condition	Rule, Policy, PolicySet
$\top$	`match` (m)	`true`(t)	`applicable`(`deny`, `permit`)
$\bot$	`notmatch` (nm)	`false`(f)	`notapplicable`(na)
I	`indeterminate` (idt)	`indeterminate`	`indeterminate`($\mathsf{i_d}$, $\mathsf{i_p}$, $\mathsf{i_{dp}}$)

B.1 XACML Components

B.1.1 Match Transformation

The evaluation of the request context in order to get attribute values in Request element that match with attribute category is as follows.

$$\texttt{eval}_M(cat, \mathcal{Q}) = \begin{cases} \texttt{error} & \text{if an error occurs during the evaluation process} \\ \{ v_1, \ldots, v_m \} & \text{if } cat(v_i) \in \{ \mathcal{A}_1, \ldots, \mathcal{A}_n \}, 1 \leq i \leq m \leq n \end{cases}$$

Match Evaluation.

$$[\![f_{MatchID}(v, cat)]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \mathtt{eval}_M(cat, \mathcal{Q}) \neq \mathtt{error} \text{ and} \\ & \quad \exists v' \in \mathtt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \top \\ \bot & \text{if } \mathtt{eval}_M(cat, \mathcal{Q}) = \{\ \} \text{ or} \\ & \quad (\mathtt{eval}_M(cat, \mathcal{Q}) \neq \mathtt{error} \text{ and} \\ & \quad\ \forall v' \in \mathtt{eval}_M(C, \mathcal{Q}) : f_{MatchID}(v, v') = \bot) \\ I & \text{if } \mathtt{eval}_M(cat, \mathcal{Q}) = \mathtt{error} \text{ or} \\ & \quad ((\forall v' \in \mathtt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') \neq \top) \text{ and} \\ & \quad\ (\exists v' \in \mathtt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = I)) \end{cases}$$

Match Transformation.

$\Pi_{\mathtt{eval}_M}$:

$evalM(cat, \mathtt{error})$	$\leftarrow \mathtt{error}.$
$evalM(cat, V)$	$\leftarrow cat(V).$
$not\_emptybag(cat)$	$\leftarrow cat(V).$
$evalM(cat, \mathsf{emptybag})$	$\leftarrow$ not $not\_emptybag(cat).$

$\Pi_{\mathcal{M}}$:

$val(\mathcal{M}, \mathsf{m})$	$\leftarrow evalM(cat, V), V \neq \mathtt{error}, f_{MatchID}(v, V, \mathtt{true}).$
$val(\mathcal{M}, \mathsf{nm})$	$\leftarrow evalM(cat, \mathsf{emptybag}).$
$val(\mathcal{M}, \mathsf{nm})$	$\leftarrow evalM(cat, V), V \neq \mathtt{error}, f_{MatchID}(v, V, \mathtt{false}),$ not $not\_false(f_{MatchID}(v, cat)).$
$not\_false(f_{MatchID}(v, cat))$	$\leftarrow evalM(cat, V), f_{MatchID}(v, V, \mathtt{true}).$
$not\_false(f_{MatchID}(v, cat))$	$\leftarrow evalM(cat, V), f_{MatchID}(v, V, \mathsf{idt}).$
$val(\mathcal{M}, \mathsf{idt})$	$\leftarrow evalM(cat, \mathtt{error}).$
$val(\mathcal{M}, \mathsf{idt})$	$\leftarrow evalM(cat, V), f_{MatchID}(v, V, \mathsf{idt}),$ $evalM(cat, V'), f_{MatchID}(v, V', \mathsf{idt}).$

Lemma B.5 *Let $\mathcal{M}$ be Match and M be an AS of $\Pi = \Pi_{\mathcal{Q}} \cup \Pi^{\mathcal{M}} \cup \Pi_{err}$. Then,*

$$\mathtt{eval}_M(cat, \mathcal{Q}) = \mathtt{error} \quad \textit{if and only if} \quad \mathtt{eval}_M(cat, \mathtt{error}) \in M\ .$$

Proof. ($\Rightarrow$) Suppose that $\mathtt{eval}_M(cat, \mathcal{Q}) = \mathtt{error}$. Then, an error occurs during evaluation. Thus $\Pi_{\mathtt{err}} = \{\, \mathtt{error} \leftarrow \top. \,\}$. Hence, $M(\mathtt{error}) = \top$. Therefore, $\mathtt{eval}_M(cat, \mathtt{error}) \in M$ by Lemma B.1.

($\Leftarrow$) Suppose that $\mathtt{eval}_M(cat, \mathtt{error}) \in M$. By Lemma B.3 there is a clause such that $\mathtt{eval}_M(cat, \mathtt{error})$ as the head and the body should be true under M. There is only one clause that $\mathtt{eval}_M(cat, \mathtt{error})$ as the head, i.e.,

$\texttt{eval}_M(cat, \texttt{error}) \leftarrow \texttt{error}.$. Thus $M(\texttt{error}) = \top$ and $\texttt{error} \in M$. By Lemma B.3 there is a clause such that error as the head and the body should be true under M. There is only one possibility that error as the head, i.e., $\{\, \texttt{error} \leftarrow \top. \,\} = \Pi_{\texttt{err}}$. Thus $\texttt{eval}_M(cat, \mathcal{Q}) = \texttt{error}$. □

Lemma B.6 *Let $\mathcal{M}$ be Match and M be an AS of $\Pi = \Pi^{\mathcal{M}}$. Then,*

$$\texttt{eval}_M(cat, \mathcal{Q}) = \{\, v_1, \ldots, v_n \,\} \quad \textit{if and only if} \quad \texttt{eval}_M(cat, v_i) \in M$$

where $1 \leq i \leq n$ and $v_i \neq \texttt{error}$.

Proof. ($\Rightarrow$) Suppose that $\texttt{eval}_M(cat, \mathcal{Q}) = \{\, v_1, \ldots, v_n \,\}$. By definition, $cat(v_i) \in \mathcal{Q}$, where $1 \leq i \leq n$. By $\mathcal{Q}$ transformation, we get $cat(v_i) \leftarrow \top. \in \Pi_{\mathcal{Q}}$. Thus, $M(cat(v_i)) = \top$). By Lemma B.1, $\texttt{eval}_M(cat, v_i) \in M$

($\Leftarrow$) Suppose that $\texttt{eval}_M(cat, v_i) \in M$. By Lemma B.3 there is a clause such that $\texttt{eval}_M(cat, v_i)$ as the head and the body should be true under M. There is only one clause that $\texttt{eval}_M(cat, v_i)$ as the head, i.e., $\texttt{eval}_M(cat, v_i) \leftarrow cat(v_i)$.. Thus $M(cat(v_i)) = \top$ and $cat(v_i) \in M$. By Lemma B.3 there is a clause such that $cat(v_i)$ as the head and the body should be true under M. There is only one possibility that $cat(v_i)$ as the head, i.e., $cat(v_i) \leftarrow \top. \in \Pi_{\mathcal{Q}}$. By $\mathcal{Q}$ transformation we get that $cat(v_i) \in \mathcal{Q}$. Thus, by definition, $\texttt{eval}_M(cat, \mathcal{Q}) = v_i$. □

Lemma B.7 *Let $\mathcal{M}$ be Match and M be an AS of $\Pi = \Pi^{\mathcal{M}}$. Then,*

$$\texttt{eval}_M(cat, \mathcal{Q}) = \emptyset \quad \textit{if and only if} \quad \texttt{eval}_M(cat, \mathsf{emptybag}) \in M$$

where $1 \leq i \leq n$.

Proof. ($\Rightarrow$) Suppose that $\texttt{eval}_M(cat, \mathcal{Q}) = \emptyset$. Then, there exists no v such that $cat(v) \in \mathcal{Q}$. Therefore, $cat(v) \leftarrow \top. \notin \Pi_{\mathcal{Q}}$. Hence, $M(cat(v)) = \bot$. Since M is a model of Π, and there exists no v such that $M(cat(v)) = \top$, then $M(not\_emptybag(cat)) = \bot$ and $M(\text{not } not\_emptybag(cat)) = \top$. Therefore, $\texttt{eval}_M(cat, \mathsf{emptybag}) = \top$. Thus, $\texttt{eval}_M(cat, \mathsf{emptybag}) \in M$

($\Leftarrow$) Suppose that $\texttt{eval}_M(cat, \mathsf{emptybag}) \in M$. By Lemma B.3 there is a clause such that $\texttt{eval}_M(cat, \mathsf{emptybag})$ as the head and the body should be true under M. There is only one clause that $\texttt{eval}_M(cat, \mathsf{emptybag})$ as the head, i.e.,

$\texttt{eval}_M(cat, \mathsf{emptybag}) \leftarrow \text{not } not\_emptybag(cat).$. Therefore, we get that $M(\text{not } not\_emptybag(cat)) = \top$. Thus, $M(not\_emptybag(cat)) = \bot$. By Lemma B.4, we get that $\forall i : M(cat(v_i)) = \bot$. Thus, there is no $\mathcal{Q}$transformation such that $cat(v_i) \leftarrow \top.$ is in $\Pi_{\mathcal{Q}}$. Hence, $\forall i : cat(v_i) \notin \mathcal{Q}$. Therefore, by definition of $\texttt{eval}_M$ we get that $\texttt{eval}_M(cat, \mathcal{Q}) = \emptyset$. □

Now we will proof the Match evaluation. The rest of the proof depends on the transformation of $f_{MatchID}/2$. We assume that the transformation of $f_{MatchID}/2$ into logic programs has the result as represented by the following conjecture.

Conjecture B.8 *Let $f_{MatchID}$ be a function used in performing the Match evaluation, v and v' be attribute values. Let $\mathcal{M} = f_{MatchID}(v, cat)$ be Match and M be an AS of $\Pi = \Pi^{\mathcal{M}}$. Then,*

$$f_{MatchID}(v, v') = X \qquad \textit{if and only if} \qquad f_{MatchID}(v, v', X) \in M \ .$$

Lemma B.9 *Let $\mathcal{M} = f_{MatchID}(v, cat)$ be Match and M be an AS of $\Pi = \Pi^{\mathcal{M}}$. Then,*

$$[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{m} \qquad \textit{if and only if} \qquad val(\mathcal{M}, \mathsf{m}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{m}$.
By Match evaluation, $\texttt{eval}_M(cat, \mathcal{Q}) \neq \texttt{error}$ and $\exists v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \top$.
By Lemma B.6, $\texttt{eval}_M(cat, v') \in M$, and $v' \neq \texttt{error}$.
By Conjecture B.8, $f_{MatchID}(v, v', \texttt{true}) \in M$.
By Lemma B.1, $val(\mathcal{M}, \mathsf{m}) \in M$ since
$\{\, \texttt{eval}_M(cat, v'), v' \neq \texttt{error}, f_{MatchID}(v, v', \texttt{true}) \,\} \subseteq M$.

($\Leftarrow$) Suppose that $val(\mathcal{M}, \mathsf{m}) \in M$.
By Lemma B.3, $M(\texttt{eval}_M(cat, v')) = \top$, $v' \neq \texttt{error}$ and
$M(f_{MatchID}(v, v', \texttt{true})) = \top$.
Let us consider $M(\texttt{eval}_M(cat, v')) = \top$. Then, $\texttt{eval}_M(cat, v') \in M$.
By Lemma B.6, we get that $v' \in \texttt{eval}_M(cat, \mathcal{Q})$.
Let us consider $M(f_{MatchID}(v, v', \texttt{true})) = \top$.
By Conjecture B.8, we get that $f_{MatchID}(v, v') = \top$.
Therefore, by Match evaluation, $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{m}$.
□

Lemma B.10 *Let $\mathcal{M} = f_{MatchID}(v, C)$ be Match and M be an AS of $\Pi = \Pi^{\mathcal{M}}$. Then,*

$$[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{nm} \qquad \textit{if and only if} \qquad val(\mathcal{M}, \mathsf{nm}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{nm}$. Based on the definition, there are two possibilities:

1. $\mathtt{eval}_M(cat, \mathcal{Q}) = \emptyset$.
   Then, by Lemma B.7, $\mathtt{eval}_M(cat, \mathsf{emptybag}) \in M$ and
   $M(\mathtt{eval}_M(cat, \mathsf{emptybag})) = \top$.
   Thus, by Lemma B.1, $val(\mathcal{M}, \mathsf{nm}) \in M$.

2. $\mathtt{eval}_M(cat, \mathcal{Q}) \neq \mathtt{error}$ and $\forall v' \in \mathtt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \bot$.
   Then, by Lemma B.6, $\mathtt{eval}_M(cat, V) \in M$ and $V \neq \mathtt{error}$.
   By Conjecture B.8, $\forall v' \in \mathtt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v', \mathtt{false}) \in M$.
   Since $\forall v' \in \mathtt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \bot$, then,
   $\nexists v' : f_{MatchID}(v, v') = \top$ or $f_{MatchID}(v, v') = \mathsf{idt}$.
   Thus, we will not have $f_{MatchID}(v, v', \mathtt{true}) \in M$ or
   $f_{MatchID}(v, v', \mathsf{idt}) \in M$.
   Therefore $M(not\_false(f_{MatchID}(v, cat))) = \bot$ since M is a model.
   Hence, $M(\text{not } not\_false(f_{MatchID}(v, cat))) = \top$.
   Since $\{\, \mathtt{eval}_M(cat, V), V \neq \mathtt{error}, \text{not } not\_false(f_{MatchID}(v, cat)) \,\} \subseteq M$.
   Thus, by Lemma B.1, $val(\mathcal{M}, \mathsf{nm}) \in M$

($\Leftarrow$) Suppose that $val(\mathcal{M}, \mathsf{nm}) \in M$. By Lemma B.3, there are two possibilities:

1. $M(\mathtt{eval}_M(cat, \mathsf{emptybag})) = \top$.
   Thus, $\mathtt{eval}_M(cat, \mathsf{emptybag}) \in M$.
   By Lemma B.7, we get that $\mathtt{eval}_M(cat, \mathcal{Q}) = \emptyset$.
   Hence, by definition, $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{nm}$.

2. $M(\mathtt{eval}_M(cat, V)) = \top, V \neq \mathtt{error}, M(\text{not } not\_false(f_{MatchID}(v, cat))) = \top$.
   Let us consider $M(\mathtt{eval}_M(cat, V)) = \top$ and $V \neq \mathtt{error}$.
   By Lemma B.6, $\mathtt{eval}_M(cat, \mathcal{Q}) = V, V \neq \mathtt{error}$.
   Let us consider $M(\text{not } not\_false(f_{MatchID}(v, cat))) = \top$.
   Therefore, $M(not\_false(f_{MatchID}(v, cat))) = \bot$.
   By Lemma B.4, we get that $\forall v_i \in \mathtt{eval}_M(cat, v_i) : M(\text{not } f_{MatchID}(v, v_i, \mathtt{false})) = \bot$).
   Moreover, $\forall v_i \in \mathtt{eval}_M(cat, v_i) : M(f_{MatchID}(v, v_i, \mathtt{false})) = \top$ since the input of $f_{MatchID}/2$ depends on $\mathtt{eval}_M/2$.

By Lemma B.6 and Conjecture B.8, $\forall v_i \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v_i) = \bot$.
Therefore, by definition $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{nm}$. □

LEMMA B.11 *Let $\mathcal{M} = f_{MatchID}(v, C)$ be Match and M be an AS of $\Pi = \Pi^{\mathcal{M}}$. Then,*

$$[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{idt} \qquad \textit{if and only if} \qquad val(\mathcal{M}, \mathsf{idt}) \in M \ .$$

PROOF. ($\Rightarrow$) Suppose that $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{idt}$. Based on the definition, there are two possibilities:

1. $\texttt{eval}_M(cat, \mathcal{Q}) = \texttt{error}$.
   Then, by Lemma B.5, $\texttt{eval}_M(cat, \texttt{error}) \in M$ and $M(\texttt{eval}_M(cat, \texttt{error})) = \top$. Thus, by Lemma B.1, $val(\mathcal{M}, \mathsf{idt}) \in M$.
2. $\exists v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') = \mathsf{idt}$ and $\forall v' \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v') \neq \top$.
   By Lemma B.6 and Conjecture B.8, $\exists v' : \texttt{eval}_M(cat, v') \in M$ and $f_{MatchID}(v, v', \mathsf{idt}) \in M$. By Lemma B.6 and Conjecture B.8, $\forall v' : \texttt{eval}_M(cat, v') \in M$ and $f_{MatchID}(v, v', \texttt{true}) \notin M$ because if $f_{MatchID}(v, v', \texttt{true}) \in M$ it will lead a contradiction. Hence, $\forall v' : M(\texttt{eval}_M(cat, v'), f_{MatchID}(v, v', \texttt{true})) = \bot$. By Lemma B.4, $M(true(f_{MatchID}(v, cat))) = \bot$. Then, $M(\text{not } true(f_{MatchID}(v, cat)) = \top$. Since $\{ \texttt{eval}_M(cat, v'), f_{MatchID}(v, v', \mathsf{idt}), \text{not } true(f_{MatchID}(v, cat)) \} \subseteq M$, hence, by Lemma B.1, $val(\mathcal{M}, \mathsf{idt}) \in M$.

($\Leftarrow$) Suppose that $val(\mathcal{M}, \mathsf{idt}) \in M$. By Lemma B.3, there are two possibilities:

1. $M(\texttt{eval}_M(cat, \texttt{error})) = \top$.
   Thus, $\texttt{eval}_M(cat, \texttt{error}) \in M$. By Lemma B.5, we get that $\texttt{eval}_M(cat, \mathcal{Q}) = \texttt{error}$. Hence, by definition, $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{idt}$.
2. $M(\texttt{eval}_M(cat, V)) = \top$, $M(f_{MatchID}(v, V, \mathsf{idt})) = \top$, and $M(\text{not } true(f_{MatchID}(v, cat))) = \top$.
   By Lemma B.6 and Conjecture B.8, $\exists v' : \texttt{eval}_M(cat, \mathcal{Q}) = v'$ and $f_{MatchID}(v, v', \mathsf{idt})$. Let us consider $M(\text{not } true(f_{MatchID}(v, cat))) = \top$. Then, $M(true(f_{MatchID}(v, cat))) = \bot$. By Lemma B.4, $\forall v_i \in \texttt{eval}_M(cat, v_i) : M(f_{MatchID}(v, v_i, \texttt{true})) = \bot$. By Lemma B.6 and Conjecture B.8, $\forall v_i \in \texttt{eval}_M(cat, \mathcal{Q}) : f_{MatchID}(v, v_i) \neq \texttt{true}$ since it leads contradiction if $f_{MatchID}(v, v_i, \texttt{true}) \notin M$ and $f_{MatchID}(v, v_i) = \texttt{true}$. Therefore, by definition $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{nm}$.

□

Proposition B.12 *Let $\Pi = \Pi_{\mathcal{Q}} \cup \Pi^{\mathcal{M}}$ be a program and M be an AS of Π. Then,*

$$[\![\mathcal{M}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{M}, V) \in M \ .$$

Proof. It follows from Lemma B.9, Lemma B.10 and Lemma B.11 since the value of V only has three possibilities, i.e., $\{ \mathsf{m}, \mathsf{nm}, \mathsf{idt} \}$. □

B.1.2 AllOf Transformation

AllOf Evaluation.

$$[\![\mathcal{A}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{M}_i]\!] = \top \\ \bot & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{M}_i]\!] = \bot \\ I & \text{otherwise} \end{cases}$$

AllOf Transformation.

$\Pi_{\mathcal{A}_{id}}$:
$val(\mathcal{A}_{id}, \mathsf{m}) \leftarrow val(\mathcal{M}_1, \mathsf{m}), \ldots, val(\mathcal{M}_n, \mathsf{m}).$
$val(\mathcal{A}_{id}, \mathsf{nm}) \leftarrow val(\mathcal{M}_i, \mathsf{nm}). \ (1 \leq i \leq n)$
$val(\mathcal{A}_{id}, \mathsf{idt}) \leftarrow \text{not } val(\mathcal{A}_{id}, \mathsf{m}), \text{not } val(\mathcal{A}_{id}, \mathsf{nm}).$

Lemma B.13 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{m} \qquad \textit{if and only if} \qquad val(\mathcal{A}, \mathsf{m}) \in M \ .$$

Proof. Let $\mathcal{A} = \bigwedge_{i=1}^{n} \mathcal{M}_i$.
($\Rightarrow$) Suppose that $[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{m}$ holds. Then, by AllOf evaluation, $\forall i : [\![\mathcal{M}_i]\!](\mathcal{Q}) = \mathsf{m}, 1 \leq i \leq n$. Based on Prop. B.12, $\forall_i : val(\mathcal{M}_i, \mathsf{m}) \in M, 1 \leq i \leq n$. Therefore, $M(val(\mathcal{M}_1, \mathsf{m}) \wedge \ldots \wedge val(\mathcal{M}_n, \mathsf{m})) = \top$. Hence, by Lemma B.1, $val(\mathcal{A}, \mathsf{m}) \in M$.
($\Leftarrow$) Suppose that $val(\mathcal{A}, \mathsf{m}) \in M$. Based on Lemma B.3, there is a rule where $val(\mathcal{A}, \mathsf{m})$ as the head and the body is true under M. Since there is only one rule in Π with $val(\mathcal{A}, \mathsf{m})$ in the head, i.e., $val(\mathcal{A}, \mathsf{m}) \leftarrow val(\mathcal{M}_1, \mathsf{m}), \ldots, val(\mathcal{M}_n, \mathsf{m})$,

we find that $M(val(\mathcal{M}_1, \mathsf{m}) \wedge \ldots \wedge val(\mathcal{M}_n, \mathsf{m})) = \top$. Therefore, $val(\mathcal{M}_i, \mathsf{m}) \in M, 1 \leq i \leq n$. Based on Prop. B.12, $[\![\mathcal{M}_i]\!](\mathcal{Q}) = \mathsf{m}, 1 \leq i \leq n$. Therefore, by AllOf evaluation, we obtain $[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{m}$. □

Lemma B.14 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{nm} \qquad \textit{if and only if} \qquad val(\mathcal{A}, \mathsf{nm}) \in M \ .$$

Proof. Let $\mathcal{A} = \bigwedge_{i=1}^{n} \mathcal{M}_i$.
($\Rightarrow$) Suppose that $[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{nm}$ holds. Then, by AllOf evaluation we have that $\exists i : [\![\mathcal{M}_i]\!](\mathcal{Q}) = \mathsf{nm}$. Based on Prop. B.12 we get that $\exists i : val(\mathcal{M}_i, \mathsf{nm}) \in M$. Thus, we get that $\exists i : M(val(\mathcal{M}_i), \mathsf{nm}) = \top$. Therefore, by Lemma B.1, $val(\mathcal{M}, \mathsf{nm}) \in M$.
($\Leftarrow$) Suppose that $val(\mathcal{A}, \mathsf{nm}) \in M$. Based on Lemma B.3 we get that there is a rule where $val(\mathcal{A}, \mathsf{nm})$ as the head and the body is true under M. Based on AllOf transformation, $\exists i : M(val(\mathcal{M}_i), \mathsf{nm}) = \top$. Therefore, $\exists i : val(\mathcal{M}_i, \mathsf{nm}) \in M$. Based on Prop. B.12 we get that $\exists i : [\![\mathcal{M}_i]\!](\mathcal{Q}) = \mathsf{nm}$. Therefore, by AllOf evaluation, we obtain $[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{nm}$. □

Lemma B.15 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{idt} \qquad \textit{if and only if} \qquad val(\mathcal{A}, \mathsf{idt}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{idt}$. Then, by AllOf evaluation, $[\![\mathcal{A}]\!](\mathcal{Q}) \neq \mathsf{m}$ and $[\![\mathcal{A}]\!](\mathcal{Q}) \neq \mathsf{nm}$. Thus, by Lemma B.13 and Lemma B.14, $val(\mathcal{A}, \mathsf{m}) \notin M$ and $val(\mathcal{A}, \mathsf{nm}) \notin M$. Hence, $M(\text{not } val(\mathcal{A}, \mathsf{m}) \wedge \text{not } val(\mathcal{A}, \mathsf{nm})) = \top$. Therefore, by Lemma B.1, $val(\mathcal{A}, \mathsf{idt}) \in M$.
($\Leftarrow$) Suppose that $val(\mathcal{A}, \mathsf{idt}) \in M$. Based on Lemma B.3, there is a rule where $val(\mathcal{A}, \mathsf{idt})$ as the head and the body is true under M. There is only one rule where $val(\mathcal{A}, \mathsf{idt})$ as the head in Π, i.e., $val(\mathcal{A}, \mathsf{idt}) \leftarrow \text{not } val(\mathcal{A}, \mathsf{m}), \text{not } val(\mathcal{A}, \mathsf{nm})$. Hence, $val(\mathcal{A}, \mathsf{m}) \notin M$ and $val(\mathcal{A}, \mathsf{nm}) \notin M$. Based on Lemma B.13 and Lemma B.14 we get that $[\![\mathcal{A}]\!](\mathcal{Q}) \neq \mathsf{m}$ and $[\![\mathcal{A}]\!](\mathcal{Q}) \neq \mathsf{nm}$. Therefore, by AllOf evaluation, we obtain $[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{idt}$. □

Proposition B.16 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{A}, V) \in M \ .$$

PROOF. It follows from Lemma B.13, Lemma B.14 and Lemma B.15 since the value of V only has three possibilities, i.e., $\{ \mathsf{m}, \mathsf{nm}, \mathsf{idt} \}$. □

B.1.3 AnyOf Transformation

AnyOf Evaluation.

$$[\![\mathcal{E}_{id}]\!](\mathcal{Q}) = \begin{cases} \top & \text{if } \exists i, 1 \leq i \leq n : [\![\mathcal{A}_i]\!] = \top \\ \bot & \text{if } \forall i, 1 \leq i \leq n : [\![\mathcal{A}_i]\!] = \bot \\ I & \text{otherwise} \end{cases}$$

AnyOf Transformation.

$\Pi_{\mathcal{E}_{id}}$:
$val(\mathcal{E}_{id}, \mathsf{m}) \quad \leftarrow val(\mathcal{A}_i, \mathsf{m}).\ (1 \leq i \leq n)$
$val(\mathcal{E}_{id}, \mathsf{nm}) \quad \leftarrow val(\mathcal{A}_1, \mathsf{nm}), \ldots, val(\mathcal{A}_n, \mathsf{nm}).$
$val(\mathcal{E}_{id}, \mathsf{idt}) \quad \leftarrow \text{not } val(\mathcal{E}_{id}, \mathsf{m}), \text{not } val(\mathcal{E}_{id}, \mathsf{nm}).$

LEMMA B.17 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{m} \qquad \textit{if and only if} \qquad val(\mathcal{A}, \mathsf{m}) \in M \ .$$

PROOF. Let $\mathcal{E} = \bigvee_{i=1}^{n} \mathcal{A}_i$
($\Rightarrow$) Suppose that $[\![\mathcal{E}]\!](\mathcal{Q}) = \mathsf{m}$ holds. Then,by AnyOf evaluation, $\exists i : [\![\mathcal{A}_i]\!](\mathcal{Q}) = \mathsf{m}, 1 \leq i \leq n$. Based on Prop. B.20, $\exists_i : val(\mathcal{A}_i, \mathsf{m}) \in M, 1 \leq i \leq n$. Thus, $\exists i : M(val(\mathcal{A}_i, \mathsf{m})) = \top$. Therefore, by Lemma B.1, $val(\mathcal{E}, \mathsf{m}) \in M$.
($\Leftarrow$) Suppose that $val(\mathcal{E}, \mathsf{m}) \in M$. Based on Lemma B.3, here is a rule where $val(\mathcal{E}, \mathsf{m})$ as the head and the body is true under M. Based on AnyOf transformation, $\exists i : M(val(\mathcal{E}_i), \mathsf{m}) = \top$. Therefore, $\exists i : val(\mathcal{E}_i, \mathsf{m}) \in M$. Based on Prop. B.20, $\exists i : [\![\mathcal{E}_i]\!](\mathcal{Q}) = \mathsf{m}$. Therefore, by AnyOf evaluation, we obtain $[\![\mathcal{E}]\!](\mathcal{Q}) = \mathsf{m}$.
□

LEMMA B.18 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{nm} \qquad \textit{if and only if} \qquad val(\mathcal{A}, \mathsf{nm}) \in M \ .$$

PROOF. Let $\mathcal{E} = \bigvee_{i=1}^{n} \mathcal{A}_i$
($\Rightarrow$) Suppose that $[\![\mathcal{E}]\!](\mathcal{Q}) = \mathsf{nm}$ holds. Then,by AnyOf evaluation, $\forall i : [\![\mathcal{A}_i]\!](\mathcal{Q}) = \mathsf{nm}$. Based on Prop. B.20, $\forall i : val(\mathcal{A}_i, \mathsf{nm}) \in M$. Thus, $M(val(\mathcal{A}_1, \mathsf{nm}) \wedge \cdots \wedge val(\mathcal{A}_n, \mathsf{nm})) = \top$. Therefore, by Lemma B.1, $val(\mathcal{E}, \mathsf{nm}) \in M$.
($\Leftarrow$) Suppose that $val(\mathcal{E}, \mathsf{nm}) \in M$. By Lemma B.3, there is a rule where $val(\mathcal{E}, \mathsf{nm})$ as the head and the body is true under M. There is only one rule in Π with $val(\mathcal{E}, \mathsf{m})$ in the head in Π, i.e., $val(\mathcal{E}, \mathsf{nm}) \leftarrow val(\mathcal{A}_1, \mathsf{nm}), \ldots, val(\mathcal{A}_n, \mathsf{nm})$. Thus, $M(val(\mathcal{A}_1, \mathsf{nm}) \wedge \ldots \wedge val(\mathcal{A}_n, \mathsf{nm})) = \top$. Therefore, $val(\mathcal{A}_i, \mathsf{nm}) \in M, 1 \leq i \leq n$. Based on Prop. B.20, $[\![\mathcal{A}_i]\!](\mathcal{Q}) = \mathsf{nm}, 1 \leq i \leq n$. Therefore, by AnyOf evaluation, we obtain $[\![\mathcal{E}]\!](\mathcal{Q}) = \mathsf{nm}$. □

LEMMA B.19 *Let $\mathcal{A}$ be an AllOf component and M be an AS of $\Pi = \Pi^{\mathcal{A}}$. Then,*

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \mathsf{idt} \qquad \textit{if and only if} \qquad val(\mathcal{A}, \mathsf{idt}) \in M \ .$$

PROOF. ($\Rightarrow$) Suppose that $[\![\mathcal{E}]\!](\mathcal{Q}) = \mathsf{idt}$. Then,by AnyOf evaluation, $[\![\mathcal{E}]\!](\mathcal{Q}) \neq \mathsf{m}$ and $[\![\mathcal{E}]\!](\mathcal{Q}) \neq \mathsf{nm}$. Thus, by Lemma B.17 and Lemma B.18, $val(\mathcal{E}, \mathsf{m}) \notin M$ and $val(\mathcal{E}, \mathsf{nm}) \notin M$. Hence, $M(\text{not } val(\mathcal{E}, \mathsf{m}) \wedge \text{not } val(\mathcal{E}, \mathsf{nm})) = \top$. By Lemma B.1, $val(\mathcal{E}, \mathsf{idt}) \in M$.
($\Leftarrow$) Suppose that $val(\mathcal{E}, \mathsf{idt}) \in M$. Based on Lemma B.3, there is a rule where $val(\mathcal{E}, \mathsf{idt})$ as the head and the body is true under M. There is only one rule in Π with $val(\mathcal{E}, \mathsf{idt})$ in the head, i.e., $val(\mathcal{E}, \mathsf{idt}) \leftarrow \text{not } val(\mathcal{E}, \mathsf{m}), \text{not } val(\mathcal{E}, \mathsf{nm})$. Hence, $val(\mathcal{E}, \mathsf{m}) \notin M$ and $val(\mathcal{E}, \mathsf{nm}) \notin M$. Based on Lemma B.17 and Lemma B.18, $[\![\mathcal{E}]\!](\mathcal{Q}) \neq \mathsf{m}$ and $[\![\mathcal{E}]\!](\mathcal{Q}) \neq \mathsf{nm}$. Therefore, by AnyOf evaluation, we obtain $[\![\mathcal{E}]\!](\mathcal{Q}) = \mathsf{idt}$. □

PROPOSITION B.20 *Let $\mathcal{E}$ be an AnyOf component and M be an AS of $\Pi = \Pi^{\mathcal{E}}$. Then,*

$$[\![\mathcal{E}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{E}, V) \in M \ .$$

PROOF. It follows from Lemma B.17, Lemma B.18 and Lemma B.19 since the value of V only has three possibilities, i.e., $\{ \mathsf{m}, \mathsf{nm}, \mathsf{idt} \}$. □

B.1.4 Target Transformation

Lemma B.21 *Let $\mathcal{T} =$ be an Target component and M be an AS of $\Pi = \Pi^{\mathcal{T}}$. Then,*

$$[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m} \qquad \textit{if and only if} \qquad val(\mathcal{T}, \mathsf{m}) \in M \ .$$

Proof. It is obvious from the definition of Target evaluation and the transformation into LP such as $val(, \mathsf{m}) \leftarrow \top.$.

Proposition B.22 *Let $\mathcal{T}$ be an Target component and M be an AS of $\Pi = \Pi^{\mathcal{T}}$. Then,*

$$[\![\mathcal{T}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{T}, V) \in M \ .$$

The proof of Prop. B.22 is similar to the proof of Prop. B.16.

B.1.5 Condition Transformation

Proposition B.23 *Let $\mathcal{C}$ be a Condition component and M be an AS of $\Pi = \Pi_{\mathcal{C}}$. Then,*

$$[\![\mathcal{C}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{C}, V) \in M \ .$$

Proof. It follows that the Condition evaluation based on the value of `eval` function, the same case in the Condition program transformation. □

B.1.6 Rule Transformation

Rule Evaluation.

$$[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \begin{cases} \top_{\mathsf{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \text{ and } E = \texttt{deny} \\ \top_{\mathsf{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \text{ and } E = \texttt{permit} \\ I_{\mathsf{d}} & \text{if } \big(([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = I) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = I\big) \text{ and } \\ & \quad E = \texttt{permit} \\ I_{\mathsf{p}} & \text{if } \big(([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = I) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = I\big) \text{ and } \\ & \quad E = \texttt{deny} \\ \bot & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \bot\big) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \end{cases}$$

Rule Transformation.

$\Pi_{\mathcal{R}_{id}}$:
$rule(\mathcal{R}_{id}, E) \quad \leftarrow \top.$
$val(\mathcal{R}_{id}, E) \quad \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{t}).$
$val(\mathcal{R}_{id}, \mathsf{i_d}) \quad \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{deny}.$
$val(\mathcal{R}_{id}, \mathsf{i_d}) \quad \leftarrow val(\mathcal{T}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{deny}.$
$val(\mathcal{R}_{id}, \mathsf{i_p}) \quad \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{permit}.$
$val(\mathcal{R}_{id}, \mathsf{i_p}) \quad \leftarrow val(\mathcal{T}, \mathsf{idt}), rule(\mathcal{R}_{id}, E), E = \texttt{permit}.$
$val(\mathcal{R}_{id}, \mathsf{na}) \quad \leftarrow val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{f}).$
$val(\mathcal{R}_{id}, \mathsf{na}) \quad \leftarrow val(\mathcal{T}, \mathsf{nm}).$

Lemma B.24 *Let* $\mathcal{R} = \texttt{rule}(\mathcal{R}_{id}) : \{ E \ ; \ \mathcal{T} \ ; \ \mathcal{C} \}$ *be a Rule component where* E *is the Rule's effect, either* `permit` *or* `deny`. *Let* M *be an AS of* $\Pi = \Pi^{\mathcal{R}}$. *Then,*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = E \qquad \textit{if and only if} \qquad val(\mathcal{R}, E) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = E$.

- $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $[\![\mathcal{C}]\!](\mathcal{Q}) = \mathsf{t}$ (by definition Rule evaluation).
- $val(\mathcal{T}, \mathsf{m}) \in M$ and $val(\mathcal{C}, \mathsf{t}) \in M$ (by Prop. B.22 and Prop. B.23).
- $val(\mathcal{R}_{id}, E) \in M$ (since $\{ val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{t}) \} \subseteq M$ and by Lemma B.1).

($\Leftarrow$) Suppose that $val(\mathcal{R}_{id}, E) \in M$.

- $M(val(\mathcal{T}, \mathsf{m}) \wedge val(\mathcal{T}, \mathsf{t})) = \top$ (by Lemma B.3).
- $val(\mathcal{T}, \mathsf{m}) \in M$ and $val(\mathcal{C}, \mathsf{t}) \in M$.
- $[\![\mathcal{M}]\!](\mathcal{Q}) = \mathsf{m}$ and $[\![\mathcal{C}]\!](\mathcal{Q}) = \mathsf{t}$ (by Prop. B.22 and Prop. B.23).
- $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = E$ (by definition Rule evaluation). □

Lemma B.25 *Let* $\mathcal{R} = \texttt{rule}(\mathcal{R}_{id}) : \{ E \ ; \ \mathcal{T} \ ; \ \mathcal{C} \}$ *be a Rule component where* E *is the Rule's effect,* $E = \texttt{deny}$. *Let* M *be an AS of* $\Pi = \Pi^{\mathcal{R}}$. *Then,*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_d} \qquad \textit{if and only if} \qquad val(\mathcal{R}, \mathsf{i_d}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$. Based on Rule evaluation definition, there are two possibilities:

1. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$, $[\![\mathcal{C}]\!] = \mathsf{idt}$, and $E = \texttt{permit}$.
   - $val(\mathcal{T}, \mathsf{m}) \in M$ and $val(\mathcal{C}, \mathsf{idt}) \in M$(by Prop. B.22 and Prop. B.23).
   - $M(rule(\mathcal{R}_{id}, E) \wedge (E = \texttt{deny})) = \top$ (since $E = \texttt{deny}$).
   - $val(\mathcal{R}_{id}, \mathsf{i_d}) \in M$ (since $\{\ val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{idt})\ \} \subseteq M$ and by Lemma B.1).
2. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $E = \texttt{permit}$.
   - $val(\mathcal{T}, \mathsf{idt}) \in M$ (by Prop. B.22).
   - $M(rule(\mathcal{R}_{id}, E) \wedge (E = \texttt{deny})) = \top$ (since $E = \texttt{deny}$).
   - $val(\mathcal{R}_{id}, \mathsf{i_d}) \in M$ (since $\{\ val(\mathcal{T}, \mathsf{idt})\ \} \subseteq M$ and by Lemma B.1).

($\Leftarrow$) Suppose that $val(\mathcal{R}_{id}, \mathsf{i_d}) \in M$. By Lemma B.3, there are two possibilities:

1. $val(\mathcal{T}, \mathsf{idt}) \in M$, $rule(\mathcal{R}_{id}, E) \in M$, and $E = \texttt{deny}$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ (by Prop. B.22).
   - $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$(by definition Rule evaluation).
2. $val(\mathcal{T}, \mathsf{m}) \in M$, $val(\mathcal{C}, \mathsf{idt}) \in M$, $rule(\mathcal{R}_{id}, E) \in M$, and $E = \texttt{deny}$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $[\![\mathcal{C}]\!](\mathcal{Q}) = \mathsf{idt}$(by Prop. B.22 and Prop. B.23).
   - $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$(by definition Rule evaluation). □

Lemma B.26 *Let $\mathcal{R} = \texttt{rule}(\mathcal{R}_{id}) : \{\ E\ ;\ \mathcal{T}\ ;\ \mathcal{C}\ \}$ be a Rule component where E is the Rule's effect, $E = \texttt{permit}$. Let M be an AS of $\Pi = \Pi^{\mathcal{R}}$. Then,*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_p} \qquad \textit{if and only if} \qquad val(\mathcal{R}, \mathsf{i_p}) \in M\ .$$

The proof of Lemma B.26 is similar like the proof Lemma B.25. The difference is that in this lemma, $E = \texttt{permit}$.

Lemma B.27 *Let $\mathcal{R} = \texttt{rule}(\mathcal{R}_{id}) : \{\ E\ ;\ \mathcal{T}\ ;\ \mathcal{C}\ \}$ be a Rule component where E is the Rule's effect, either* `permit` *or* `deny`*. Let M be an AS of $\Pi = \Pi^{\mathcal{R}}$. Then,*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{na} \qquad \textit{if and only if} \qquad val(\mathcal{R}, \mathsf{na}) \in M\ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \mathsf{na}$. Based on Rule evaluation definition, there are two possibilities:

1. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$, $[\![\mathcal{C}]\!] = \mathsf{f}$.
   - $val(\mathcal{T}, \mathsf{m}) \in M$ and $val(\mathcal{C}, \mathsf{f}) \in M$(by Prop. B.22 and Prop. B.23).
   - $val(\mathcal{R}_{id}, \mathsf{na}) \in M$ (since $\{\ val(\mathcal{T}, \mathsf{m}), val(\mathcal{C}, \mathsf{f})\ \} \subseteq M$ and by Lemma B.1).
2. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{nm}$.
   - $val(\mathcal{T}, \mathsf{nm}) \in M$ (by Prop. B.22).
   - $val(\mathcal{R}_{id}, \mathsf{nm}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $val(\mathcal{R}_{id}, \mathsf{nm}) \in M$. By Lemma B.3, there are two possibilities:

1. $val(\mathcal{T}, \mathsf{m}) \in M$, $val(\mathcal{C}, \mathsf{f}) \in M$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $[\![\mathcal{C}]\!](\mathcal{Q}) = \mathsf{f}$ (by Prop. B.22 and Prop. B.23).
   - $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \mathsf{na}$(by definition Rule evaluation).
2. $val(\mathcal{T}, \mathsf{nm}) \in M$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{nm}$ (by Prop. B.22).
   - $[\![\mathcal{R}_{id}]\!](\mathcal{Q}) = \mathsf{nm}$(by definition Rule evaluation). □

Proposition B.28 *Let $\mathcal{R}$ be a Rule component where E is the Rule's effect, either* `permit` *or* `deny`*. Let M be an AS of $\Pi = \Pi^{\mathcal{R}}$. Then,*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{R}, V) \in M \ .$$

Proof. It follows from Lemma B.24, Lemma B.25, Lemma B.26, and Lemma B.27 since the value of V only has five possibilities, i.e., $\{\ \mathsf{permit}, \mathsf{deny}, \mathsf{i_d}, \mathsf{i_p}, \mathsf{na}\ \}$. □

B.2 Combining Algorithms Transformation

B.2.1 Permit-Overrides Combining Algorithm

Permit-Overrides Combining Algorithm Evaluation. Let $\langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6 and let $\mathbf{S} = \{\ s_1, \ldots, s_n\ \}$ be a set of policy

values from $\mathbb{S}$. We define the *permit-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) = \bigsqcup_{\mathtt{po}} \mathbf{S}$$

Permit-Overrides Combining Algorithm Transformation.

$\Pi_{\mathtt{po}}$:

$$
\begin{array}{ll}
\mathit{algo}(\mathtt{po}, P, \mathtt{permit}) & \leftarrow \mathit{decision\_of}(P, R, \mathtt{permit}). \\
\mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \mathit{decision\_of}(P, R, \mathsf{i}_{\mathsf{dp}}). \\
\mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \\
 & \quad \mathit{decision\_of}(P, R_1, \mathsf{i}_{\mathsf{p}}), \mathit{decision\_of}(P, R_2, \mathtt{deny}). \\
\mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \\
 & \quad \mathit{decision\_of}(P, R_1, \mathsf{i}_{\mathsf{p}}), \mathit{decision\_of}(P, R_2, \mathsf{i}_{\mathsf{d}}). \\
\mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{p}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}), \\
 & \quad \mathit{decision\_of}(P, R, \mathsf{i}_{\mathsf{p}}). \\
\mathit{algo}(\mathtt{po}, P, \mathtt{deny}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}), \\
 & \quad \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{p}}), \mathit{decision\_of}(P, R, \mathtt{deny}). \\
\mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{d}}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}), \\
 & \quad \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{p}}), \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{deny}), \\
 & \quad \mathit{decision\_of}(P, R, \mathsf{i}_{\mathsf{d}}). \\
\mathit{algo}(\mathtt{po}, P, \mathsf{na}) & \leftarrow \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{permit}), \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}), \\
 & \quad \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{p}}), \text{not } \mathit{algo}(\mathtt{po}, P, \mathtt{deny}), \\
 & \quad \text{not } \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{d}}).
\end{array}
$$

We use PO for an abbreviation of permit-overrides combining algorithm.

Lemma B.29 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathtt{po} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathtt{po}}$. *Then,*

$$\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathtt{permit} \qquad \textit{if and only if} \qquad \mathit{algo}(\mathtt{po}, \mathcal{P}_{id}, \mathtt{permit}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathtt{permit}$.

- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \mathtt{permit}$ (by PO evaluation).
- $\mathit{val}(\mathcal{R}_i, \mathtt{permit}) \in M$ (by Prop. B.28).
- There is a clause in Π $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_i, \mathtt{permit}) \leftarrow \mathit{val}(\mathcal{R}_i, \mathtt{permit})$. (by Policy transformation).

- $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_i, \texttt{permit}) \in M$ (by Lemma B.1).
- $\mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $\mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \in M$.

- $M(\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}, \texttt{permit})) = \top$ (by Lemma B.3).
- $M(\mathit{val}(\mathcal{R}, \texttt{permit})) = \top$ (by Lemma B.3). Therefore, $\mathit{val}(\mathcal{R}, \texttt{permit}) \in M$.
- $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{permit}$ (by Prop. B.28).
- $\bigoplus_{\texttt{po}}(\mathbb{R}) = \texttt{permit}$ (by PO evaluation). □

Lemma B.30 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{\, \texttt{po} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{po}}$. *Then,*

$$\bigoplus_{\texttt{po}}(\mathbb{R}) = \mathsf{i}_{\mathsf{dp}} \qquad \textit{if and only if} \qquad \mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\texttt{po}}(\mathbb{R}) = \mathsf{i}_{\mathsf{dp}}$. Based on PO evaluation definition, there are three possibilities:

1. $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \texttt{permit}$ and $\exists j : [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{i}_{\mathsf{dp}}$.
   - $\forall i : \mathit{val}(\mathcal{R}_i, \texttt{permit}) \notin M$ and $\exists j : \mathit{val}(\mathcal{R}_j, \mathsf{i}_{\mathsf{dp}}) \in M$ (by Prop. B.28).
   - $\mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \notin M$ by Lemma B.29 since if it is in M, there exists a Rule $\mathcal{R}$ in the Policy $\mathcal{P}_{id}$sequence such that $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{permit}$.
   - $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_j, \mathsf{i}_{\mathsf{dp}}) \in M$ (by Policy transformation and Lemma B.1).
   - $\mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \in M$ (by Lemma B.1).
2. $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \texttt{permit}$ and $\exists j : [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{i}_{\mathsf{p}}$ and $\exists j' : [\![\mathcal{R}_{j'}]\!](\mathcal{Q}) = \texttt{deny}$.
   - $\forall i : \mathit{val}(\mathcal{R}_i, \texttt{permit}) \notin M$ and
     $\exists j : \mathit{val}(\mathcal{R}_j, \mathsf{i}_{\mathsf{p}}) \in M$ and $\exists j : \mathit{val}(\mathcal{R}_{j'}, \texttt{deny}) \in M$ (by Prop. B.28).
   - $\mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \notin M$ by Lemma B.29 since if it is in M, there exists a Rule $\mathcal{R}$ in the Policy $\mathcal{P}_{id}$sequence such that $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{permit}$.
   - $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_j, \mathsf{i}_{\mathsf{p}}) \in M$ and $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_{j'}, \texttt{deny}) \in M$ (by Policy transformation and Lemma B.1).
   - $\mathit{algo}(\texttt{po}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \in M$ (by Lemma B.1).

3. $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \texttt{permit}$ and $\exists j : [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{i_p}$ and $\exists j' : [\![\mathcal{R}_{|'}]\!](\mathcal{Q}) = \mathsf{i_d}$.
   - $\forall i : val(\mathcal{R}_i, \texttt{permit}) \notin M$ and $\exists j : val(\mathcal{R}_j, \mathsf{i_p}) \in M$ and $\exists j : val(\mathcal{R}_{j'}, \mathsf{i_d}) \in M$ (by Prop. B.28).
   - $algo(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \notin M$ by Lemma B.29 since if it is in M, there exists a Rule $\mathcal{R}$ in the Policy $\mathcal{P}_{id}$sequence such that $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{permit}$.
   - $decision\_of(\mathcal{P}_{id}, \mathcal{R}_j, \mathsf{i_p}) \in M$ and $decision\_of(\mathcal{P}_{id}, \mathcal{R}_{j'}, \mathsf{i_d}) \in M$ (by Policy transformation and Lemma B.1).
   - $algo(\texttt{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\texttt{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$. By Lemma B.3, there are three possibilities:

1. $M(\text{not } algo(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \wedge decision\_of(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_{dp}})) = \top$.
   - Then, $algo(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \notin M$ and $decision\_of(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_{dp}}) \in M$.
   - $\bigoplus_{\texttt{po}}(\mathbb{R}) \neq \texttt{permit}$ (by Lemma B.29 since there is a contradiction if $\bigoplus_{\texttt{po}}(\mathbb{R}) = \texttt{permit}$).
   - $M(val(\mathcal{R}, \mathsf{i_{dp}})) = \top$ (by Policy transformation and Lemma B.3).
   - Then, $val(\mathcal{R}, \mathsf{i_p}) \in M$.
   - $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \texttt{permit}$ since $\bigoplus_{\texttt{po}}(\mathbb{R}) \neq \texttt{permit}$ (by PO evaluation).
   - $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_{dp}}$ (by Prop. B.28).
   - $\bigoplus_{\texttt{po}}(\mathbb{R}) = \mathsf{i_{dp}}$ (by PO evaluation).
2. $M(\text{not } algo(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \wedge decision\_of(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_p}) \wedge decision\_of(\mathcal{P}_{id}, \mathcal{R}', \texttt{deny})) = \top$.
   - Then, $algo(\texttt{po}, \mathcal{P}_{id}, \texttt{permit}) \notin M$, $decision\_of(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_p}) \in M$, and $decision\_of(\mathcal{P}_{id}, \mathcal{R}', \texttt{deny}) \in M$.
   - $\bigoplus_{\texttt{po}}(\mathbb{R}) \neq \texttt{permit}$ (by Lemma B.29 since there is a contradiction if $\bigoplus_{\texttt{po}}(\mathbb{R}) = \texttt{permit}$).
   - $M(val(\mathcal{R}, \mathsf{i_p})) = \top$ and $M(val(\mathcal{R}, \texttt{deny})) = \top$ (by Policy transformation and Lemma B.3).
   - Then, $val(\mathcal{R}, \mathsf{i_p}) \in M$ and $val(\mathcal{R}, \texttt{deny}) \in M$.
   - $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \texttt{permit}$ since $\bigoplus_{\texttt{po}}(\mathbb{R}) \neq \texttt{permit}$ (by PO evaluation).
   - $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_p}$ and $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{deny}$ (by Prop. B.28).
   - $\bigoplus_{\texttt{po}}(\mathbb{R}) = \mathsf{i_{dp}}$ (by PO evaluation).

3. $M(\text{not } \mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathtt{permit}) \wedge \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_p}) \wedge \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}', \mathsf{i_d})) = \top$.
   - Then, $\mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathtt{permit}) \notin M$, $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_p}) \in M$, and $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}', \mathsf{i_d}) \in M$.
   - $\bigoplus_{\mathsf{po}}(\mathbb{R}) \neq \mathtt{permit}$ (by Lemma B.29 since there is a contradiction if $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathtt{permit}$).
   - $M(\mathit{val}(\mathcal{R}, \mathsf{i_p})) = \top$ and $M(\mathit{val}(\mathcal{R}, \mathsf{i_d})) = \top$ (by Policy transformation and Lemma B.3).
   - Then, $\mathit{val}(\mathcal{R}, \mathsf{i_p}) \in M$ and $\mathit{val}(\mathcal{R}, \mathsf{i_d}) \in M$.
   - $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \mathtt{permit}$ since $\bigoplus_{\mathsf{po}}(\mathbb{R}) \neq \mathtt{permit}$ (by PO evaluation).
   - $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_p}$ and $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by Prop. B.28).
   - $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathsf{i_{dp}}$ (by PO evaluation). □

Lemma B.31 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathsf{po} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathsf{po}}$. *Then,*

$$\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathsf{i_p} \qquad \textit{if and only if} \qquad \mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathsf{i_p}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathsf{i_p}$.

- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \mathsf{i_p}$ and $\forall j : [\![\mathcal{R}_j]\!](\mathcal{Q}) \neq \mathsf{i_p} \Rightarrow [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{na}$ (by PO evaluation).
- $\exists i : \mathit{val}(\mathcal{R}_j, \mathsf{i_p}) \in M$ (by Prop. B.28).
- $\mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathtt{permit}) \notin M$ by Lemma B.29 since if it is in M, there exists a Rule $\mathcal{R}$ in the Policy $\mathcal{P}_{id}$sequence such that $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathtt{permit}$.
- $\mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \notin M$ by Lemma B.29 since if it is in M there exists a Rule $\mathcal{R}$ in the Policy $\mathcal{P}_{id}$sequence such that $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_{dp}}$, and $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{d}$ or $\mathsf{i_d}$.
- $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_i, \mathsf{i_p}) \in M$ (by Policy transformation and Lemma B.1).
- $\mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathsf{i_p}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $\mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathsf{i_p}) \in M$.

- $M(\text{not } \mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathtt{permit}) \wedge \text{not } \mathit{algo}(\mathsf{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \wedge \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_p})) = \top$ (by Lemma B.3).

- Then, $algo(\mathtt{po}, \mathcal{P}_{id}, \mathtt{permit}) \notin M$, $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \notin M$ and $decision\_of(\mathcal{P}_{id}, \mathcal{R}, \mathsf{i_p}) \in M$.
- $\bigoplus_{\mathtt{po}}(\mathbb{R}) \neq \mathtt{permit}$ and $\bigoplus_{\mathtt{po}}(\mathbb{R}) \neq \mathsf{i_{dp}}$ (by Lemma B.29 and Lemma B.30 since there is a contradiction if $\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathtt{permit}$ and $\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathsf{i_{dp}}$).
- $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \neq \mathtt{p}$ and $[\![\mathcal{R}_i]\!](\mathcal{Q}) \neq (\mathsf{i_{dp}}$ or $\mathtt{d}$ or $\mathsf{i_d})$ (by PO evaluation)
- $M(val(\mathcal{R}, \mathsf{i_p})) = \top$ (by Policy transformation and Lemma B.3).
- Then, $val(\mathcal{R}, \mathsf{i_p}) \in M$.
- $[\![\mathcal{R}]\!](\mathcal{Q}) = \mathsf{i_p}$ (by Prop. B.28).
- $\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathsf{i_p}$ (by PO evaluation). □

Lemma B.32 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathtt{po} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathtt{po}}$. *Then,*

$$\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathtt{deny} \qquad \textit{if and only if} \qquad algo(\mathtt{po}, \mathcal{P}_{id}, \mathtt{deny}) \in M \ .$$

Lemma B.33 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathtt{po} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathtt{po}}$. *Then,*

$$\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathtt{deny} \qquad \textit{if and only if} \qquad algo(\mathtt{po}, \mathcal{P}_{id}, \mathtt{deny}) \in M \ .$$

Lemma B.34 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathtt{po} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathtt{po}}$. *Then,*

$$\bigoplus_{\mathtt{po}}(\mathbb{R}) = \mathsf{na} \qquad \textit{if and only if} \qquad algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{na}) \in M \ .$$

The proofs of Lemma B.32, Lemma B.33, and Lemma B.34 are similar to the proof of Lemma B.31.

Proposition B.35 *Let* M *be an AS of* $\Pi = \Pi^{\mathtt{po}}$. *Then,*

$$\bigoplus_{\mathtt{po}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad algo(\mathtt{po}, \mathcal{P}_{id}, V) \in M \ .$$

Proof. It follows from Lemma B.29, Lemma B.30, Lemma B.31, Lemma B.32, Lemma B.33 and Lemma B.34 since the value of V only has six possibilities, i.e., $\{\, \mathtt{permit}, \mathtt{deny}, \mathsf{i_p}, \mathsf{i_d}, \mathsf{i_{dp}}, \mathsf{na} \,\}$. □

B.2.2 Deny-Overrides Combining Algorithm

Proposition B.36 *Let M be an AS of $\Pi = \Pi$. Then,*

$$\bigoplus_{\mathtt{po}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad \mathit{algo}(\mathtt{po}, \mathcal{P}_{id}, V) \in M \ .$$

The proof of Prop. B.36 is similar to the proof of Prop. B.35 since deny-overrides combining algorithm is the mirror of permit-overrides combining algorithm.

B.2.3 Legacy-Permit-Overrides Combining Algorithm

Legacy Permit-Overrides Combining Algorithm Evaluation. Let $\mathbb{S} = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values from V_6. We define the *legacy permit-overrides combining algorithm under* V_6 as follows:

$$\bigoplus_{\mathtt{lpo}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathsf{dp}} & \text{if } \bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) \in \{\ I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}}\ \} \\ \bigoplus_{\mathtt{po}}^{V_6}(\mathbb{S}) & \text{otherwise} \end{cases}$$

Permit-Overrides Combining Algorithm Transformation.

$$\begin{array}{ll} \mathit{algo}(\mathtt{lpo}, P, \mathsf{i}_{\mathsf{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{p}}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{i}_{\mathsf{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{d}}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{i}_{\mathsf{dp}}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{i}_{\mathsf{dp}}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathtt{permit}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathtt{permit}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathtt{deny}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathtt{deny}) \\ \mathit{algo}(\mathtt{lpo}, P, \mathsf{na}) & \leftarrow \mathit{algo}(\mathtt{po}, P, \mathsf{na}) \end{array}$$

We use LPO for an abbreviation of legacy permit-overrides combining algorithm.

Lemma B.37 *Let $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\ \mathtt{lpo}\ ;\ \mathcal{T}\ ;\ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle\ \}$ be a Policy component and M be an AS of $\Pi = \Pi^{\mathtt{lpo}}$. Then,*

$$\bigoplus_{\mathtt{lpo}}(\mathbb{R}) = \mathsf{i}_{\mathsf{dp}} \qquad \textit{if and only if} \qquad \mathit{algo}(\mathtt{lpo}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathtt{lpo}}(\mathbb{R}) = \mathsf{i}_{\mathsf{dp}}$.

- $\bigoplus_{\mathsf{po}}(\mathbb{R}) = (\mathsf{i_{dp}}$ or $\mathsf{i_p}$ or $\mathsf{i_d})$ (by LPO evaluation).
- $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$ or $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_p}) \in M$ or $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$ (by Prop. B.35).
- $algo(\mathtt{lpo}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\mathtt{lpo}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$.

- $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$ or $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$ or $algo(\mathtt{po}, \mathcal{P}_{id}, \mathsf{i_p}) \in M$ (by Lemma B.3).
- $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathsf{i_{dp}}$ or $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathsf{i_d}$ or $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathsf{i_p}$ (by Prop. B.35).
- $\bigoplus_{\mathtt{lpo}}(\mathbb{R}) = \mathsf{i_{dp}}$ (by LPO evaluation). □

Lemma B.38 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathtt{lpo} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathtt{lpo}}$. *Then,*

$$\bigoplus_{\mathtt{lpo}}(\mathbb{R}) = \mathtt{permit} \qquad \textit{if and only if} \qquad algo(\mathtt{lpo}, \mathcal{P}_{id}, \mathtt{permit}) \in M \;.$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathtt{lpo}}(\mathbb{R}) = \mathtt{permit}$.

- $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathtt{permit}$ (by LPO evaluation).
- $algo(\mathtt{po}, \mathcal{P}_{id}, \mathtt{permit}) \in M$ (by Prop. B.35).
- $algo(\mathtt{lpo}, \mathcal{P}_{id}, \mathtt{permit}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\mathtt{lpo}, \mathcal{P}_{id}, \mathtt{permit}) \in M$.

- $algo(\mathtt{po}, \mathcal{P}_{id}, \mathtt{permit}) \in M$ (by Lemma B.3).
- $\bigoplus_{\mathsf{po}}(\mathbb{R}) = \mathtt{permit}$ (by Prop. B.35).
- $\bigoplus_{\mathtt{lpo}}(\mathbb{R}) = \mathtt{permit}$ (by LPO evaluation). □

Lemma B.39 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{\, \texttt{lpo} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{lpo}}$*. Then,*

$$\bigoplus_{\texttt{lpo}}(\mathbb{R}) = \texttt{deny} \qquad \textit{if and only if} \qquad \textit{algo}(\texttt{lpo}, \mathcal{P}_{id}, \texttt{deny}) \in M \;\; .$$

Lemma B.40 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{\, \texttt{lpo} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{lpo}}$*. Then,*

$$\bigoplus_{\texttt{lpo}}(\mathbb{R}) = \texttt{na} \qquad \textit{if and only if} \qquad \textit{algo}(\texttt{lpo}, \mathcal{P}_{id}, \texttt{na}) \in M \;\; .$$

The proofs of Lemma B.39 and Lemma B.40 are similar to the proof of Lemma B.38

Proposition B.41 *Let* M *be an AS of* $\Pi = \Pi^{\texttt{lpo}}$*. Then,*

$$\bigoplus_{\texttt{po}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad \textit{algo}(\texttt{po}, \mathcal{P}_{id}, V) \in M \;\; .$$

Proof. It follows from Lemma B.37, Lemma B.38, Lemma B.39, Lemma B.40 since the value of V only has four possibilities, i.e., $\{\, \texttt{permit}, \texttt{deny}, \texttt{i}_{\texttt{dp}}, \texttt{na} \,\}$. □

B.2.4 Legacy Deny-Overrides Combining Algorithm

Proposition B.42 *Let* M *be an AS of* $\Pi = \Pi^{\texttt{ldo}}$*. Then,*

$$\bigoplus_{\texttt{po}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad \textit{algo}(\texttt{po}, \mathcal{P}_{id}, V) \in M \;\; .$$

The proof of Prop. B.42 is similar to the proof of Prop. B.41 since legacy deny-overrides combining algorithm is the mirror of legacy permit-overrides combining algorithm.

B.2.5 Permit-Unless-Deny Combining Algorithm

Permit-Unless-Deny Combining Algorithm Evaluation.

$$\bigoplus_{\texttt{pud}}^{V_6}(\mathbb{S}) = \begin{cases} \top_{\texttt{d}} & \text{if } \exists i \in \{\, 1, \ldots, n \,\} : s_i = \top_{\texttt{d}} \\ \top_{\texttt{p}} & \text{otherwise} \end{cases}$$

Permit-Unless-Deny Combining Algorithm Transformation.

$$
\begin{array}{ll}
\Pi_{\texttt{pud}}: & \\
\mathit{algo}(\texttt{pud}, P, \texttt{deny}) & \leftarrow \mathit{decision\_of}(P, R, \texttt{deny}). \\
\mathit{algo}(\texttt{pud}, P, \texttt{permit}) & \leftarrow \text{not } \mathit{algo}(\texttt{dup}, P, \texttt{deny}).
\end{array}
$$

We use PUD for an abbreviation of permit-unless-deny combining algorithm.

Lemma B.43 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{\, \texttt{pud} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{pud}}$. *Then,*

$$\bigoplus_{\texttt{pud}}(\mathbb{R}) = \texttt{deny} \qquad \textit{if and only if} \qquad \mathit{algo}(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny}) \in M \;\; .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\texttt{pud}}(\mathbb{R}) = \texttt{deny}$.

- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \texttt{deny}$ (by PUD evaluation).
- $\mathit{val}(\mathcal{R}_i, \texttt{deny}) \in M$ (by Prop. B.28).
- There is a clause in Π $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_i, \texttt{deny}) \leftarrow \mathit{val}(\mathcal{R}_i, \texttt{deny})$. (by-Policy transformation).
- $\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_i, \texttt{deny}) \in M$ (by Lemma B.1).
- $\mathit{algo}(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $\mathit{algo}(\texttt{pud}, \mathcal{P}_{id}, \texttt{permit}) \in M$.

- $M(\mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}, \texttt{deny})) = \top$ (by Lemma B.3).
- $M(\mathit{val}(\mathcal{R}, \texttt{deny})) = \top$ (by Lemma B.3). Therefore, $\mathit{val}(\mathcal{R}, \texttt{deny}) \in M$.
- $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{deny}$ (by Prop. B.28).
- $\bigoplus_{\texttt{pud}}(\mathbb{R}) = \texttt{deny}$ (by PO evaluation). □

Lemma B.44 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{\, \texttt{pud} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{pud}}$. *Then,*

$$\bigoplus_{\texttt{pud}}(\mathbb{R}) = \texttt{permit} \qquad \textit{if and only if} \qquad \mathit{algo}(\texttt{pud}, \mathcal{P}_{id}, \texttt{permit}) \in M \;\; .$$

PROOF. ($\Rightarrow$) Suppose that $\bigoplus_{\texttt{pud}}(\mathbb{R}) = \texttt{permit}$.

- $\forall i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = V$ and $V \neq \texttt{deny}$ (by PUD evaluation).
- $val(\mathcal{R}_i, V) \in M$ and $V \neq \texttt{deny}$ (by Prop. B.28).
- Then, $\bigoplus_{\texttt{pud}}(\mathbb{R}) \neq \texttt{deny}$.
- $algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny}) \notin M$ (by Lemma B.43 since there will be a contradiction if $algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny}) \in M$).
- Then, $M(\text{not } algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny})) = \top$
- $algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{permit}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{permit}) \in M$.

- $M(\text{not } algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny})) = \top$ (by Lemma B.3).
- Then, $M(algo(\texttt{pud}, \mathcal{P}_{id}, \texttt{deny})) = \bot$.
- $\forall i : M(decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, \texttt{deny})) = \bot$ (by Lemma B.4).
- $\forall i : M(val(\mathcal{R}_i, \texttt{deny})) = \bot$ (by Lemma B.4). Therefore, $val(\mathcal{R}, \texttt{deny}) \notin M$.
- $\nexists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \texttt{deny}$ (by Prop. B.28).
- $\bigoplus_{\texttt{pud}}(\mathbb{R}) = \texttt{permit}$ (by PUD evaluation). □

PROPOSITION B.45 *Let M be an AS of $\Pi = \Pi^{\texttt{pud}}$. Then,*

$$\bigoplus_{\texttt{pud}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad algo(\texttt{pud}, \mathcal{P}_{id}, V) \in M \ .$$

PROOF. It follows from Lemma B.43 and Lemma B.44 since the value of V only has two possibilities, i.e., $\{ \texttt{permit}, \texttt{deny}, \texttt{i}_{\texttt{dp}}, \texttt{na} \}$. □

B.2.6 Deny-Unless-Permit Combining Algorithm

Proposition B.46 *Let M be an AS of $\Pi = \Pi^{\mathtt{pud}}$. Then,*

$$\bigoplus_{\mathtt{dup}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad \mathit{algo}(\mathtt{dup}, \mathcal{P}_{id}, V) \in M \ .$$

The proof of Prop. B.46 is similar to the proof of Prop. B.45 since deny-unless-permit combining algorithm is the mirror of permit-unless-deny combining algorithm.

B.2.7 First-Applicable Combining Algorithm

First-Applicable Combining Algorithm Evaluation.

$$\bigoplus_{\mathtt{fa}}^{V_6}(\mathbb{S}) = \begin{cases} I_{\mathtt{dp}} & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ I_{\mathtt{d}}, I_{\mathtt{p}}, I_{\mathtt{dp}} \} \wedge \forall j : (j < i) \Rightarrow (s_j = \bot) \\ s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{ \top_{\mathtt{p}}, \top_{\mathtt{d}} \} \wedge \forall j : (j < i) \Rightarrow (s_j = \bot) \\ \bot & \text{otherwise} \end{cases}$$

First-Applicable Combining Algorithm Transformation.

$\Pi_{\mathtt{fa}}$:

$$\begin{array}{ll} \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) & \leftarrow \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_1, V), V \neq \mathtt{na}. \\ \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) & \leftarrow \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_1, \mathtt{na}), \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_2, V), V \neq \mathtt{na}. \\ & \vdots \\ \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) & \leftarrow \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_1, \mathtt{na}), \ldots, \mathit{decision\_of}(\mathcal{P}_{id}, \mathcal{R}_{n-1}, \mathtt{na}), \\ & \quad \mathit{decision\_of}(\mathcal{P}_{id}, R_n, V). \end{array}$$

We use FA for an abbreviation of first-applicable combining algorithm.

Proposition B.47 *Let $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{ \mathit{comb}_{id} \ ; \ \mathcal{T} \ ; \ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \}$ be a Policy component and M be an AS of $\Pi = \Pi^{\mathtt{fa}}$. Then,*

$$\bigoplus_{\mathtt{fa}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad \mathit{algo}(\mathtt{fa}, \mathcal{P}_{id}, V) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathtt{fa}}(\mathbb{R}) = V$.

- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = V$ and $V \neq \mathsf{na}$ and $\forall j : j < i \Rightarrow [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{na}$ (by FA evaluation).
- $\exists i : val(\mathcal{R}_i, V) \in M$ where $V \neq \mathsf{na}$ and $\forall j : j < i \Rightarrow val(\mathcal{R}_j, \mathsf{na}) \in M$ (by Prop. B.28).
- $decision\_of(\mathcal{P}, \mathcal{R}_i, V) \in M$ and $\forall j : j < i \Rightarrow decision\_of(\mathcal{P}, \mathcal{R}_j, \mathsf{na}) \in M$ (by Policy transformation and since M is a minimal model for Π).
- $algo(\mathtt{fa}, \mathcal{P}, V) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\mathtt{fa}, \mathcal{P}, V) \in M$.

- Based on Lemma B.3, there is a clause in $\mathcal{P}$ where $algo(\mathtt{fa}, \mathcal{P}, V)$ as the head and the body is true under M. There are several clauses in $\mathcal{P}$ where $algo(\mathtt{fa}, \mathcal{P}, V)$ as the head. We can see that in each rule the body contains $\exists i : decision\_of(\mathcal{P}, \mathcal{R}_i, V), V \neq \mathsf{na}$ and $\forall j : j < i \Rightarrow decision\_of(\mathcal{P}, \mathcal{R}_j, \mathsf{na})$.
- $\exists i : decision\_of(\mathcal{P}, \mathcal{R}_i, V) \in M$ and $\forall j : j < i \Rightarrow decision\_of(\mathcal{P}, \mathcal{R}_j, \mathsf{na}) \in M$.
- $\exists i : M(val(\mathcal{R}_i, V)) = \top$ and $\forall j < i : M(val(\mathcal{R}_j, \mathsf{na})) = \top$ (by Lemma B.3).
- Then, $\exists i : val(\mathcal{R}_i, V) \in M$ and $\forall j : j < i \Rightarrow val(\mathcal{R}_j, \mathsf{na}) \in M$.
- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = V$ and $\forall j : j < i \Rightarrow [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{na}$ (by Prop. B.28).
- $\bigoplus_{\mathtt{fa}}(\mathbb{R}) = V$ (by FA evaluation). □

B.2.8 Only-One-Applicable Combining Algorithm

Only-One-Applicable Algorithm Evaluation.

$$\bigoplus_{\mathtt{ooa}}^{V_6}(\mathbb{S}) = \begin{cases} s_i & \text{if } \exists i \in \{1,\ldots,n\} : s_i \in \{\top_{\mathsf{d}}, \top_{\mathsf{p}}\} \wedge \\ & \quad \forall j \in \{1,\ldots,n\} : j \neq i \Rightarrow s_j = \bot \\ I_{\mathsf{dp}} & \text{if } \big(\exists i \in \{1,\ldots,n\} : s_i \in \{I_{\mathsf{d}}, I_{\mathsf{p}}, I_{\mathsf{dp}}\}\big) \vee \\ & \quad \big(\exists i,j \in \{1,\ldots,n\} : i \neq j \wedge s_i, s_j \in \{\top_{\mathsf{d}}, \top_{\mathsf{p}}\}\big) \\ \bot & \text{otherwise} \end{cases}$$

Only-One-Applicable Algorithm Transformation.

Π_{ooa} :

$$
\begin{array}{ll}
algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) & \leftarrow decision\_of(\mathcal{P}_{id}, \mathcal{R}_1, V_1), decision\_of(\mathcal{P}_{id}, \mathcal{R}_2, V_2), \mathcal{R}_1 \neq \mathcal{R}_2, \\
& \quad applicable\_value(V_1), applicable\_value(V_2). \\
algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) & \leftarrow decision\_of(\mathcal{P}_{id}, \mathcal{R}, V), idt\_value(V). \\
algo(\mathsf{ooa}, \mathcal{P}_{id}, V) & \leftarrow \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}), \\
& \quad decision\_of(\mathcal{P}_{id}, \mathcal{R}, applicable\_value), applicable\_value(V). \\
algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{na}) & \leftarrow \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}), \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathtt{deny}), \\
& \quad \text{not } algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathtt{permit}).
\end{array}
$$

We use OOA for an abbreviation of permit-overrides combining algorithm.

Lemma B.48 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathsf{ooa} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathsf{ooa}}$. *Then,*

$$\bigoplus_{\mathsf{ooa}}(\mathbb{R}) = \mathsf{i_{dp}} \qquad \textit{if and only if} \qquad algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathsf{ooa}}(\mathbb{R}) = \mathsf{i_{dp}}$. Based on OOA evaluation definition, there are two possibilities:

1. $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = (\mathsf{i_d}$ or $\mathsf{i_p}$ or $\mathsf{i_{dp}})$.
   - $\exists i : val(\mathcal{R}_i, V) \in M$ and $idt\_value(V) \in M$ (by Prop. B.28).
   - There is a clause in Π $decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, V) \leftarrow val(\mathcal{R}_i, V)$. (by Policy transformation).
   - $decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, V) \in M$ and $idt\_value(V) \in M$ (by Lemma B.1).
   - $algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$ (by Lemma B.1).
2. $\exists i, j : i \neq j$ and $semantics\mathcal{R}_i(\mathcal{Q}) = \mathtt{permit}$ or $\mathtt{deny}$ and $semantics\mathcal{R}_j(\mathcal{Q}) = \mathtt{permit}$ or $\mathtt{deny}$ and $\mathcal{R}_i \neq \mathcal{R}_j$.
   - $\exists i, j : val(\mathcal{R}_i, V_1) \in M$ and $val(\mathcal{R}_j, V_2)$ and $M(applicable\_value(V_1)) = \top$ and $M(applicable\_value(V_2)) = \top$ and $\mathcal{R}_i \neq \mathcal{R}_j$ (by Prop. B.28).
   - $decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, V_1) \in M$ and $decision\_of(\mathcal{P}_{id}, \mathcal{R}_j, V_j) \in M$ and $M(applicable\_value(V_1)) = \top$ and $M(applicable\_value(V_2)) = \top$ and $\mathcal{R}_i \neq \mathcal{R}_j$ (by Policy transformation and by Lemma B.1).
   - $algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\mathsf{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$. By Lemma B.3, there two three possibilities:

1. $M(decision\_of(\mathcal{P}_{id}, \mathcal{R}_1, V_1) \wedge decision\_of(\mathcal{P}_{id}, \mathcal{R}_2, V_2) \wedge \mathcal{R}_1 \neq \mathcal{R}_2 \wedge applicable\_value(V_1) \wedge applicable\_value(V_2)) = \top$.
   - $M(val(\mathcal{R}_2, V_1)) = \top$, $M(val(\mathcal{R}_2, V_2)) = \top$, $\mathcal{R}_1 \neq \mathcal{R}_2$, $M(applicable\_value(V_1)) = \top$, and $M(applicable\_value(V_2)) = \top$(by Lemma B.3). Therefore, $val(\mathcal{R}1, V_1) \in M$, $val(\mathcal{R}_2, V_2) \in M$.
   - $[\![\mathcal{R}_1]\!](\mathcal{Q}) = V_1$, $[\![\mathcal{R}_2]\!](\mathcal{Q}) = V_2$, $\mathcal{R}_1 \neq \mathcal{R}_2$, $V_1, V_2 \in \{\, \mathtt{deny}, \mathtt{permit} \,\}$ (by Prop. B.28).
   - $\bigoplus_{\mathsf{ooa}}(\mathbb{R}) = \mathsf{i_{dp}}$ (by OOA evaluation).
2. $M(decision\_of(\mathcal{P}_{id}, \mathcal{R}, V) \wedge idt\_value(V)) = \top$.
   - $M(val(\mathcal{R}, V)) = \top$ and $M(idt\_value(V)) = \top$ (by Lemma B.3). Therefore, $val(\mathcal{R}, V) \in M$.
   - $[\![\mathcal{R}]\!](\mathcal{Q}) = V$, and $V \in \{\, \mathsf{i_d}, \mathsf{i_p}, \mathsf{i_{dp}} \,\}$ (by Prop. B.28).
   - $\bigoplus_{\mathsf{ooa}}(\mathbb{R}) = \mathsf{i_{dp}}$ (by OOA evaluation). □

Lemma B.49 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, \mathtt{ooa} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\mathtt{ooa}}$. *Then,*

$$\bigoplus_{\mathtt{ooa}}(\mathbb{R}) = \mathit{Deny} \qquad \text{if and only if} \qquad algo(\mathtt{ooa}, \mathcal{P}_{id}, \mathit{Deny}) \in M \;\; .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\mathsf{ooa}}(\mathbb{R}) = \mathtt{deny}$.

- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \mathtt{deny}$, $\forall j : i \neq j \Rightarrow [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathtt{na}$ (by OOA evaluation).
- $\exists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \mathtt{deny}$, and $\bigoplus_{\mathsf{ooa}}(\mathbb{R}) \neq \mathsf{i_{dp}}$.
- $\exists i : val(\mathcal{R}_i, \mathtt{deny}) \in M$ and $algo(\mathtt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \notin M$ (by Prop. B.28 and by Lemma B.48 since there is a contradiction if $algo(\mathtt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \in M$).
- There is a clause in Π $decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, \mathtt{deny}) \leftarrow val(\mathcal{R}_i, \mathtt{deny})$. (by Policy transformation).
- $decision\_of(\mathcal{P}_{id}, \mathcal{R}_i, \mathtt{deny}) \in M$ and $applicable\_value(\mathtt{deny}) \in M$ and $M(\text{not } algo(\mathtt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}})) = \top$ (by Lemma B.1).
- $algo(\mathtt{ooa}, \mathcal{P}_{id}, \mathtt{deny}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\mathtt{ooa}, \mathcal{P}_{id}, \mathtt{deny}) \in M$.

- $M(\text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \wedge decision\_of(\mathcal{P}_{id}, \mathcal{R}, \texttt{deny}) \wedge applicable\_value(\texttt{deny})) = \top$ (by Lemma B.3).
- Then, $M(algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}})) = \bot$, $decision\_of(\mathcal{P}_{id}, \mathcal{R}, \texttt{deny}) \in M$.
- $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \mathsf{i_{dp}}$ (by Lemma B.48 since there will be a contradiction if $\bigoplus_{\texttt{ooa}}(\mathbb{R}) = \mathsf{i_{dp}}$). $M(val(\mathcal{R}, \texttt{deny})) = \top$ (by Lemma B.3). Therefore, $val(\mathcal{R}, \texttt{deny}) \in M$.
- $\nexists i : [\![\mathcal{R}_i]\!](\mathcal{Q}) \in \{ \mathsf{i_d}, \mathsf{i_p}, \mathsf{i_{dp}} \}$ and $\nexists i, j : i \neq j \wedge [\![\mathcal{R}_i]\!](\mathcal{Q}), [\![\mathcal{R}_j]\!](\mathcal{Q}) \in \{ \texttt{deny}, \texttt{permit} \}$ and $[\![\mathcal{R}]\!](\mathcal{Q}) = \texttt{deny}$ (by Prop. B.28).
- $\forall i : [\![\mathcal{R}_i]\!] \notin \{ \mathsf{i_d}, \mathsf{i_p}, \mathsf{i_{dp}} \}$. We have a particular i such that $[\![\mathcal{R}_i]\!](\mathcal{Q}) = \texttt{deny}$. Thus, for the second equation, we conclude that $\forall j : i \neq j \Rightarrow [\![\mathcal{R}_j]\!](\mathcal{Q}) \notin \{ \texttt{deny}, \texttt{permit} \}$.
- Hence, the only possible value of $\forall j : [\![\mathcal{R}_j]\!](\mathcal{Q}) = \mathsf{na}$.
- $\bigoplus_{\texttt{ooa}}(\mathbb{R}) = \texttt{deny}$ (by OOA evaluation). □

Lemma B.50 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{ \texttt{ooa} ; \mathcal{T} ; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{ooa}}$. *Then,*

$$\bigoplus_{\texttt{ooa}}(\mathbb{R}) = \mathit{Permit} \quad \textit{if and only if} \quad algo(\texttt{ooa}, \mathcal{P}_{id}, \mathit{Permit}) \in M \ .$$

The proof of Lemma B.50 is similar to the proof Lemma B.49.

Lemma B.51 *Let* $\mathcal{P} = \texttt{policy}(\mathcal{P}_{id}) : \{ \texttt{ooa} ; \mathcal{T} ; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \}$ *be a Policy component and* M *be an AS of* $\Pi = \Pi^{\texttt{ooa}}$. *Then,*

$$\bigoplus_{\texttt{ooa}}(\mathbb{R}) = \mathsf{na} \quad \textit{if and only if} \quad algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{na}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $\bigoplus_{\texttt{ooa}}(\mathbb{R}) = \mathsf{na}$.

- $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \mathsf{i_{dp}}$, $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \texttt{deny}$, and $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \texttt{permit}$ (by OOA evaluation).
- $algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) \notin M$, $algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) \notin M$, and $algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{permit}) \notin M$ (by Lemma B.48, Lemma B.49 and Lemma B.50),
- $M(algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i_{dp}}) = \bot)$, $M(algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) = \bot)$, $M(algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) = \bot)$.

- $M(\text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) = \top)$, $M(\text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) = \top)$, $M(\text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) = \top)$.
- $algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{na}) \in M$ (by Lemma B.1).

($\Leftarrow$) Suppose that $algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{na}) \in M$.

- $M(\text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \wedge \text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) \wedge \text{not } algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{permit})) = \top$ (by Lemma B.3).
- $algo(\texttt{ooa}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \notin M$,$algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{deny}) \notin M$ and $algo(\texttt{ooa}, \mathcal{P}_{id}, \texttt{permit}) \notin M$.
- $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \mathsf{i}_{\mathsf{dp}}$, $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \texttt{deny}$ and $\bigoplus_{\texttt{ooa}}(\mathbb{R}) \neq \texttt{permit}$ (by Lemma B.48, Lemma B.49 and Lemma B.50).
- $\bigoplus_{\texttt{ooa}}(\mathbb{R}) = \mathsf{na}$ (by OOA equation). □

Proposition B.52 *Let M be an AS of $\Pi = \Pi^{\texttt{ooa}}$. Then,*

$$\bigoplus_{\texttt{ooa}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad algo(\texttt{ooa}, \mathcal{P}_{id}, V) \in M \ .$$

Proof. It follows from Lemma B.48, Lemma B.49, Lemma B.50, and Lemma B.51 since the value of V only has four possibilities, i.e., $\{ \texttt{permit}, \texttt{deny}, \mathsf{i}_{\mathsf{dp}}, \mathsf{na} \}$. □

B.2.9 All Combining Algorithms

Proposition B.53 *Let M be an AS of $\Pi = \Pi^{comb_{id}}$. Then,*

$$\bigoplus_{comb_{id}}(\mathbb{R}) = V \qquad \textit{if and only if} \qquad algo(comb_{id}, \mathcal{P}_{id}, V) \in M \ .$$

Proof. It follows from Prop. B.35, Prop. B.36, Prop. B.41, Prop. B.42, Prop. B.45, Prop. B.46, Prop. B.47, Prop. B.52. □

B.3 Policy and PolicySet Transformation

Policy Evaluation.

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \begin{cases} \top_{\mathsf{d}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \top_{\mathsf{d}} \\ \top_{\mathsf{p}} & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \top_{\mathsf{p}} \\ I_{\mathsf{dp}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = I_{\mathsf{dp}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \big(I \text{ or } I_{\mathsf{dp}}\big)\big) \\ I_{\mathsf{d}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = I_{\mathsf{d}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \big(\top_{\mathsf{d}} \text{ or } I_{\mathsf{d}}\big)\big) \\ I_{\mathsf{p}} & \text{if } \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = I_{\mathsf{p}}\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \big(\top_{\mathsf{p}} \text{ or } I_{\mathsf{p}}\big)\big) \\ \bot & \text{if } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \bot\big) \text{ or} \\ & \quad \big([\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \bot\big) \end{cases}$$

where $\mathbb{R} = \langle [\![\mathcal{R}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{R}_n]\!](\mathcal{Q}) \rangle$.

Policy Transformation.

$\Pi_{\mathcal{P}_{id}}$:

$val(\mathcal{P}_{id}, \mathtt{permit}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathtt{permit}).$
$val(\mathcal{P}_{id}, \mathtt{deny}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{idt}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{dp}}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{d}}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{d}}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{d}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{d}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{d}}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{p}}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{p}}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{p}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathtt{permit}).$
$val(\mathcal{P}_{id}, \mathsf{i}_{\mathsf{p}}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i}_{\mathsf{p}}).$
$val(\mathcal{P}_{id}, \mathsf{na}) \leftarrow val(\mathcal{T}, \mathsf{nm}).$
$val(\mathcal{P}_{id}, \mathsf{na}) \leftarrow val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{na}).$
$val(\mathcal{P}_{id}, \mathsf{na}) \leftarrow val(\mathcal{T}, \mathsf{idt}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{na}).$

Lemma B.54 *Let $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\ comb_{id}\ ;\ \mathcal{T}\ ;\ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle\ \}$ be a Policy component. Let M be an AS of $\Pi = \Pi^{\mathcal{P}}$. Then,*

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathtt{deny} \qquad \textit{if and only if} \qquad val(\mathcal{P}_{id}, \mathtt{deny}) \in M\ .$$

PROOF. ($\Rightarrow$) Suppose that $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathtt{deny}$.

- $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathtt{deny}$ (by definition of Policy evaluation).
- $val(\mathcal{T}, \mathsf{m}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny}) \in M$ (by Prop. B.22 and Prop. B.53).
- $val(\mathcal{P}_{id}, \mathtt{deny}) \in M$ (since $\{\, val(\mathcal{T}, \mathsf{m}), algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny}) \,\} \subseteq M$ and by Lemma B.1).

($\Leftarrow$) Suppose that $val(\mathcal{P}_{id}, \mathtt{deny}) \in M$.

- $M(val(\mathcal{T}, \mathsf{m}) \wedge algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny})) = \top$ (by Lemma B.3).
- Then, $val(\mathcal{T}, \mathsf{m}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny}) \in M$.
- $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathtt{deny}$ (by Prop. B.22 and Prop. B.53).
- $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathtt{deny}$ (by definition Policy evaluation).

□

LEMMA B.55 *Let $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, comb_{id} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ be a Policy component. Let M be an AS of $\Pi = \Pi^{\mathcal{P}}$. Then,*

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathtt{permit} \qquad \textit{if and only if} \qquad val(\mathcal{P}_{id}, \mathtt{permit}) \in M \ .$$

The proof of Lemma B.55 is similar like the proof Lemma B.54. The difference is that in this lemma, $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathtt{permit}$.

LEMMA B.56 *Let $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\, comb_{id} \;;\; \mathcal{T} \;;\; \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle \,\}$ be a Policy component. Let M be an AS of $\Pi = \Pi^{\mathcal{P}}$. Then,*

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d} \qquad \textit{if and only if} \qquad val(\mathcal{P}_{id}, \mathsf{i_d}) \in M \ .$$

PROOF. ($\Rightarrow$) Suppose that $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$. Based on Policy evaluation definition, there are three possibilities:

1. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathsf{i_d}$.

- $val(\mathcal{T}, \mathsf{m}) \in M$ and $algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$ (by Prop. B.22 and Prop. B.53).
- $val(\mathcal{P}_{id}, \mathsf{i_d}) \in M$ (since $\{ val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{deny}) \} \subseteq M$ and by Lemma B.1).

2. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \mathsf{deny}$.
   - $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{deny}) \in M$ (by Prop. B.22 and Prop. B.53).
   - $val(\mathcal{P}_{id}, \mathsf{i_d}) \in M$ (since $\{ val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{deny}) \} \subseteq M$ and by Lemma B.1).
3. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \mathsf{i_d}$.
   - $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$ (by Prop. B.22 and Prop. B.53).
   - $val(\mathcal{P}_{id}, \mathsf{i_d}) \in M$ (since $\{ val(\mathcal{T}, \mathsf{m}), algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{deny}) \} \subseteq M$ and by Lemma B.1).

($\Leftarrow$) By Lemma B.3, there are three possibilities:

1. $M(val(\mathcal{T}, \mathsf{m}) \wedge algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d})) = \top$.
   - Then, $val(\mathcal{T}, \mathsf{m}) \in M$ and $algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $\bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \mathsf{i_d}$ (by Prop. B.22 and Prop. B.53).
   - $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by definition Policy evaluation).
2. $M(val(\mathcal{T}, \mathsf{idt}) \wedge algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{deny})) = \top$.
   - Then, $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{deny}) \in M$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \mathsf{deny}$ (by Prop. B.22 and Prop. B.53).
   - $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by definition Policy evaluation).
3. $M(val(\mathcal{T}, \mathsf{idt}) \wedge algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d})) = \top$.
   - Then, $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\text{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$.
   - $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\text{comb}_{id}}(\mathbb{R}) = \mathsf{i_d}$ (by Prop. B.22 and Prop. B.53).
   - $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by definition Policy evaluation).

□

Lemma B.57 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\ comb_{id}\ ;\ \mathcal{T}\ ;\ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle\ \}$ *be a Policy component. Let* M *be an AS of* $\Pi = \Pi^{\mathcal{P}}$*. Then,*

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_p} \qquad \textit{if and only if} \qquad val(\mathcal{P}_{id}, \mathsf{i_p}) \in M \ .$$

Lemma B.58 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\ comb_{id}\ ;\ \mathcal{T}\ ;\ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle\ \}$ *be a Policy component. Let* M *be an AS of* $\Pi = \Pi^{\mathcal{P}}$*. Then,*

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_{dp}} \qquad \textit{if and only if} \qquad val(\mathcal{P}_{id}, \mathsf{i_{dp}}) \in M \ .$$

The proofs of Lemma B.57 and Lemma B.58 are similar to the proof of Lemma B.56.

Lemma B.59 *Let* $\mathcal{P} = \mathtt{policy}(\mathcal{P}_{id}) : \{\ comb_{id}\ ;\ \mathcal{T}\ ;\ \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle\ \}$ *be a Policy component. Let* M *be an AS of* $\Pi = \Pi^{\mathcal{P}}$*. Then,*

$$[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{na} \qquad \textit{if and only if} \qquad val(\mathcal{P}_{id}, \mathsf{na}) \in M \ .$$

Proof. ($\Rightarrow$) Suppose that $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$. Based on Policy evaluation definition, there are three possibilities:

1. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{nm}$.
   - $val(\mathcal{T}, \mathsf{nm}) \in M$ (by Prop. B.22).
   - $val(\mathcal{P}_{id}, \mathsf{na}) \in M$ (by Lemma B.1).
2. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathsf{na}$.
   - $val(\mathcal{T}, \mathsf{m}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{na}) \in M$ (by Prop. B.22 and Prop. B.53).
   - $val(\mathcal{P}_{id}, \mathsf{na}) \in M$ (since $\{\ val(\mathcal{T}, \mathsf{m}), algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{na})\ \} \subseteq M$ and by Lemma B.1).
3. $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathsf{na}$.
   - $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{na}) \in M$ (by Prop. B.22 and Prop. B.53).
   - $val(\mathcal{P}_{id}, \mathsf{na}) \in M$ (since $\{\ val(\mathcal{T}, \mathsf{idt}), algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{na})\ \} \subseteq M$ and by Lemma B.1).

($\Leftarrow$) By Lemma B.3, there are three possibilities:

1. $M(val(\mathcal{T}, \mathsf{m}) \wedge algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d})) = \top$.

- Then, $val(\mathcal{T}, \mathsf{m}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$.
- $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{m}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathsf{i_d}$ (by Prop. B.22 and Prop. B.53).
- $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by definition Policy evaluation).

2. $M(val(\mathcal{T}, \mathsf{idt}) \wedge algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny})) = \top$.

- Then, $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathtt{deny}) \in M$.
- $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathtt{deny}$ (by Prop. B.22 and Prop. B.53).
- $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by definition Policy evaluation).

3. $M(val(\mathcal{T}, \mathsf{idt}) \wedge algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d})) = \top$.

- Then, $val(\mathcal{T}, \mathsf{idt}) \in M$ and $algo(\mathrm{comb}_{id}, \mathcal{P}_{id}, \mathsf{i_d}) \in M$.
- $[\![\mathcal{T}]\!](\mathcal{Q}) = \mathsf{idt}$ and $\bigoplus_{\mathrm{comb}_{id}}(\mathbb{R}) = \mathsf{i_d}$ (by Prop. B.22 and Prop. B.53).
- $[\![\mathcal{P}_{id}]\!](\mathcal{Q}) = \mathsf{i_d}$ (by definition Policy evaluation).

□

Proposition B.60 *Let $\mathcal{P}$ be a Policy component and M be an AS of $\Pi = \Pi^{\mathcal{P}}$. Then,*

$$[\![\mathcal{P}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{P}, V) \in M \ .$$

Proof. It follows from Lemma B.54, Lemma B.55, Lemma B.56, Lemma B.57, Lemma B.58, Lemma B.59 since the value of V only has six possibilities, i.e., $\{\, \mathtt{permit}, \mathtt{deny}, \mathsf{i_d}, \mathsf{i_p}, \mathsf{i_{dp}}, \mathsf{na} \,\}$. □

Proposition B.61 *Let $\mathcal{PS}$ be a PolicySet component and M be an AS of $\Pi = \Pi^{\mathcal{PS}}$. Then,*

$$[\![\mathcal{PS}]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(\mathcal{PS}, V) \in M \ .$$

The proof of Prop. B.61 is similar to the proof of Prop. B.60 since the PolicySet is constructed in the same way as Policy.

B.4 XACML-ASP

Corollary B.62 *Let* $\Pi = \Pi_{\mathcal{Q}} \cup \Pi_{XACML}$ *be a program obtained by merging Request transformation program* $\Pi_{\mathcal{Q}}$ *and all XACML components transformation programs* Π_{XACML}*. Let* M *be an AS of* Π*. Then,*

$$[\![X]\!](\mathcal{Q}) = V \qquad \textit{if and only if} \qquad val(X, V) \in M$$

where X *is an XACML component.*

Proof. It follows from Prop. B.12, Prop. B.16, Prop. B.20, Prop. B.22, Prop. B.23, Prop. B.28, Prop. B.60, and Prop. B.61. □

Appendix C

Listing of Access Control Policies in Smart Grid

Listing C.1: Access Control Policies for Smart Grid

```
policyset(smartgrid):{ ooa;
        target(smartgrid):{null};
        < eha, data, powerline, billing, pricelist >
}
```

C.1 Access Control Policies for EHA

1. SM can *turn on* an EHA and it is mandatory to write a log when the EHA is started.
2. SM can *turn off* an EHA and it is mandatory to write a log when the EHA is ended.
3. Through SM, the SP can do a particular action to the corresponding EHA, for example, *update* a new software, and it is mandatory to write the description on the action in the log.

We assume that there is only one request that is allowed each time. However, there might be possibility two policies are applicable, for example, SM received

an request from SP in order to turn off a particular EHA. Thus, policy 1 and policy 3 are both applicable. Hence, in this policy set, we prefer to set "first-applicable" combining operator.

Listing C.2: Access Control Policies for EHA

```
policy(eha):{fa;
  target(eha):{
    anyof(ehaany):{
      allof(ehaall):{
        string-equal(eha, resource-type)
      }}};
  < eha1, eha2, eha3 >
}

rule(eha1):{permit;
  target(eha1):{
    anyof(eha1-any):{
      allof(eha1-all):{
        string-equal(sm, subject), string-equal(turn-on, action)
      }}};
  condition(eha1):{true}
}

rule(eha2):{permit;
  target(eha2):{
    anyof(eha2-any):{
      allof(eha2-all):{
        string-equal(sm, subject),
        string-equal(turn-off, action)
      }}};
  condition(eha2):{true}
}

rule(eha3):{permit;
  target(eha3):{
    anyof(eha3-any):{
      allof(eha3-all):{
        string-equal(sp, subject),
        string-equal(update, action)
      }}};
  condition(eha2):{(eha-id(X) /\ sp-id(Y) /\ eha-sp(X,Y))}
}
```

C.2 Access Control Policies for Data

1. SM can *write* the aggregated data consumption and technical data.

2. DSO can *read* the aggregated data consumption from SM.

3. HC can *read* the detailed consumption data from SM.

4. SP can *read* the data about a particular EHA.

In order to protect data secrecy, we prefer to deny all access request unless we are sure that the requestor has access to it. Hence, we use "deny-unless-permit" combining algorithm.

Listing C.3: Access Control Policies for Data

```
policy(data):{ dup;
        target(data):{ null };
        <data1, data2, data3, data4, data5>
}

rule(data1):{ permit;
  target(data1):{
        anyof(data1-any):{
                        allof(data1-all):{
                                string-equal(sm, subject),
                                string-equal(read, action),
                                anyURI-equal(urn:example:DSO:
                                    aggregated-data-record,
                                    target-namespace),
                                string-equal(aggregate-data-
                                    record, resource-type)
                        }
                }
        };
        condition(data1):{true}
}

rule(data2):{ permit;
  target(data2):{
        anyof(data2-any):{
                        allof(data2-all):{
                                string-equal(sm, subject),
                                string-equal(write, action),
                                anyURI-equal(urn:example:DSO:
                                    technical-data-record,
                                    target-namespace),
                                string-equal(technical-data-
                                    record, resource-type)
                        }
                }
        };
        condition(data2):{true}
}

rule(data3):{ permit;
  target(data3):{
        anyof(data3-any):{
                        allof(data3-all):{
```

```
38                                      string-equal(dso, subject),
39                                      string-equal(write, action),
40                                      anyURI-equal(urn:example:DSO:
                                            aggregate-data-record,
                                            target-namespace),
41                                      string-equal(aggregate-data-
                                            record, resource-type)
42                              }}};
43          condition(data3):{true}
44  }
45
46  rule(data4):{ permit;
47    target(data4):{
48          anyof(data4-any):{
49                              allof(data4-all):{
50                                      string-equal(customer, subject),
51                                      string-equal(read, action),
52                                      anyURI-equal(urn:example:SM:
                                            detailed-data-record, target
                                            -namespace),
53                                      string-equal(detailed-data-
                                            record, resource-type)
54                              }}};
55          condition(data4):{(subject-id(X) /\ customer-id(X))}
56  }
57
58  rule(data5):{ permit;
59    target(data5):{
60          anyof(data5-any):{
61                              allof(data5-all):{
62                                      string-equal(read, action),
63                                      anyURI-equal(urn:example:SM:
                                            technical-data-record,
                                            target-namespace),
64                                      string-equal(technical-data-
                                            record, resource-type)
65                              }
66                  }
67          };
68          condition(data5):{(subject-id(X) /\ related-EHA(X,Y) /\
                technical-data-id(Y))}
69  }
```

C.3 Access Control Policies for Power Line

1. SM can *connect* to the power line.
2. SM can *disconnect* from the power line.
3. DSO can *connect* SM to the power line.

4. DSO can *disconnect* SM from the power line if the customer has not paid the bill or there is a technical safety reason.

5. EHA can *connect* the electricity through SM and it is mandatory to write a log when the EHA is started.

6. EHA can *disconnect* the electricity through SM and it is mandatory to write a log when the EHA is ended.

In order to make sure that the supply power is always available, any permitted access to the power system should be allowed. In this case, we use "permit-unless-deny" combining operator.

Listing C.4: Access Control Policies for Power Line

```
policy(powerline):{ pud;
        target(powerline):{
        anyof(powerline-any):{
                        allof(powerline-all):{string-equal(
                            electric-power, resource-type)}
                }
  };
        <powerline1, powerline2, powerline3, powerline4,
            powerline5, powerline6>
}

rule(powerline1):{ permit;
  target(powerline1):{
        anyof(powerline1-any):{
                        allof(powerline1-all):{
                                string-equal(sm, subject),
                                    string-equal(connect, action
                                    )
                        }
                }
        };
        condition(powerline1):{true}
}

rule(powerline2):{ permit;
  target(powerline2):{
        anyof(powerline2-any):{
                        allof(powerline2-all):{
                                string-equal(sm, subject),
                                    string-equal(disconnect,
                                    action)
                        }
                }
        };
        condition(powerline2):{true}
}

```

```
32 rule(powerline3):{ permit;
33   target(powerline3):{
34         anyof(powerline3-any):{
35                         allof(powerline3-all):{
36                                 string-equal(dso, subject),
                                       string-equal(connect, action
                                       )
37                         }
38                 }
39         };
40         condition(powerline3):{true}
41 }
42
43 rule(powerline4):{ permit;
44   target(powerline4):{
45         anyof(powerline4-any):{
46                         allof(powerline4-all):{
47                                 string-equal(dso, subject),
                                       string-equal(disconnect,
                                       action)
48                         }
49                 }
50         };
51         condition(powerline4):{technical-issue(critical) \/
               billing-status(not-yet-paid)}
52 }
53
54 rule(powerline5):{ permit;
55   target(powerline5):{
56         anyof(powerline5-any):{
57                         allof(powerline5-all):{
58                                 string-equal(eha, subject),
                                       string-equal(connect, action
                                       )
59                         }
60                 }
61         };
62         condition(powerline5):{true}
63 }
64
65 rule(powerline6):{ permit;
66   target(powerline6):{
67         anyof(powerline6-any):{
68                         allof(powerline6-all):{
69                                 string-equal(eha, subject),
                                       string-equal(disconnect,
                                       action)
70                         }
71                 }
72         };
73         condition(powerline6):{true}
74 }
75
76
77 rule(powerline1):{ permit;
```

```
78    target(powerline1):{
79          anyof(powerline1-any):{
80                              allof(powerline1-all1):{
81                                      string-equal(sm, subject),
                                            string-equal(connect, action
                                            )
82                              },
83                              allof(powerline1-all2):{
84                                      string-equal(eha, subject),
                                            string-equal(connect, action
                                            )
85                              }
86                  }
87          };
88          condition(powerline1):{true}
89  }
90
91  rule(powerline2):{ permit;
92    target(powerline2):{
93          anyof(powerline2-any1):{
94                              allof(powerline2-all):{
95                                      string-equal(sm, subject),
                                            string-equal(disconnect,
                                            action)
96                              }
97                  },
98                  anyof(powerline2-any2):{
99                              allof(powerline2-all):{
100                                     string-equal(eha, subject),
                                            string-equal(disconnect,
                                            action)
101                             }
102                 }
103         };
104         condition(powerline2):{true}
105 }
```

C.4 Access Control Policies for Billing Statement

1. HC can *read* the billing statement from the FI.
2. HC can *pay* the billing to the FI.
3. DSO can *send* bill to the FI.
4. DSO can *read* the payment bill from FI.
5. FI can *update* the payment status after the payment transaction succeed

We would like that the bill is always paid and the access to the billing statement is always granted. However, since secrecy is important, we only give permission to the trustworthy requestor. Thus, we use "permit-overrides" combining operator.

Listing C.5: Access Control Policies for Billing Statement

```
policy(billing):{ po;
        target(billing):{
        anyof(billing-any):{
                        allof(billing-all):{
                                anyURI-equal(urn:example:FI:
                                    billing-statement-record,
                                    target-namespace), string-
                                    equal(billing-statement,
                                    resource-type)}
                }
  };
        <billing1, billing2, billing3, billing4, billing5>
}

rule(billing1):{ permit;
  target(billing1):{
        anyof(billing1-any):{
                        allof(billing1-all):{
                                string-equal(customer, subject),
                                     string-equal(read, action)
                        }
                }
        };
        condition(billing1):{(customer-id(X) /\ fi-id(F) /\
            customer-list(F, X) /\ billing-record-id(F, X))}
}

rule(billing2):{ permit;
  target(billing2):{
        anyof(billing2-any):{
                        allof(billing2-all):{
                                string-equal(customer, subject),
                                     string-equal(pay, action)
                        }
                }
        };
        condition(billing2):{(customer-id(X) /\ fi-id(F) /\
            customer-list(F, X) /\ billing-record-id(F, X))}
}

rule(billing3):{ permit;
  target(billing3):{
        anyof(billing3-any):{
                        allof(billing3-all):{
                                string-equal(dso, subject),
                                    string-equal(send, action)
                        }
                }
```

```
40          };
41          condition(billing3):{(dso-id(X) /\ fi-id(F) /\ dso-list(
                F, X) /\ billing-record-id(F, X) /\ billing-status(X
                , new))}
42  }
43
44  rule(billing4):{ permit;
45    target(billing4):{
46          anyof(billing4-any):{
47                              allof(billing4-all):{
48                                      string-equal(dso, subject),
                                            string-equal(read, action)
49                              }
50                  }
51          };
52          condition(billing4):{(dso-id(X) /\ fi-id(F) /\ dso-list(
                F, X) /\ billing-record-id(F, X))}
53  }
54
55  rule(billing5):{ permit;
56    target(billing5):{
57          anyof(billing5-any):{
58                              allof(billing5-all):{
59                                      string-equal(fi, subject),
                                            string-equal(update, action)
60                              }
61                  }
62          };
63          condition(billing4):{(customer-id(X) /\ fi-id(F) /\
                customer-list(F, X) /\ billing-record-id(F, X) /\
                payment-status(X, paid))}
64  }
```

C.5 Access Control Policies for Price List

1. DSO can *update* the price list to the SM.
2. HC through SM can *see* the price list.
3. SM can *access* the price list in order to make a plan.

The price list is available to public and not secret. Thus, we use "permit-unless-deny" combining operator.

Listing C.6: Access Control Policies for Price List

```
1  policy(pricelist):{ pud;
2    target(pricelist):{
3      anyof(pricelist-any):{
```

```
4         allof(pricelist-all):{
5           anyURI-equal(urn:example:DSO:pricelist, target-namespace
                ), string-equal(pricelist, resource-type)}
6         }
7     };
8           <pricelist1, pricelist2>
9   }
10
11  rule(pricelist1):{ permit;
12    target(pricelist1):{
13          anyof(pricelist1-any):{
14                              allof(pricelist1-all):{
15                                      string-equal(dso, subject),
                                            string-equal(update, action)
16                              }
17                  }
18          };
19          condition(pricelist1):{true}
20  }
21
22  rule(pricelist2):{ permit;
23    target(pricelist2):{
24          anyof(pricelist2-any):{
25                              allof(pricelist2-all):{
26                                      string-equal(customer, subject),
                                             string-equal(read, action)
27                              }
28                  }
29          };
30          condition(pricelist1):{(dso-id(X) /\ SM-id(SM) /\ dso-SN
                (X, SM) /\ subject-id(S) /\ customer-id(S))}
31  }
```

Bibliography

[ABGG12] Teresa Alsinet, Ramón Béjar, Lluis Godo, and Francesc Guitart. Using answer set programming for an scalable implementation of defeasible argumentation. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 1016–1021, 2012.

[AHK+03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Technical report, http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/, 2003.

[AHLM10a] G.-J. Ahn, H. Hu, J. Lee, and Y. Meng. Reasoning about XACML policy descriptions in answer set programming (preliminary report). In *NMR'10*, 2010.

[AHLM10b] G.-J. Ahn, H. Hu, J. Lee, and Y. Meng. Representing and reasoning about web access control policies. In *COMPSAC*. IEEE Computer Society, 2010.

[And72] James P. Anderson. Computer security technology planning study. Technical report esd-tr-73-51, Electronic System Division/AFSC, Bedford, MA, October 1972.

[AvE82] Krzysztof R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM (JACM)*, 29(3):841–862, 1982.

[Bar03] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge, 2003.

[BBA07] Zinaida Benenson, Peter M. Cholewinski B, and Felix C. Freiling A. Vulnerabilities and attacks in wireless sensor networks. In J. Lopez and J. Zhou, editors, *Wireless Sensors Networks Security (IOS Press).*, 2007.

[BBC+06a] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Philipp Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, Claus von Riegen, Daniel Roth, Jeffrey Schlimmer, Chris Sharp, John Shewchuk, Asi Vedamuthu, Ümit Yalçinalp, and David Orchard. Web services policy framework (WS Policy). Technical report, http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf, March 2006.

[BBC+06b] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Hiroshi Maruyama, Anthony Nadalin, David Orchard, Hemma Prafullchandra, Claus von Riegen, Daniel Roth, Jeffrey Schlimmer, Chris Sharp, John Shewchuk, Asi Vedamuthu, and Ü. Web services policy attachment (WS PolicyAttachment). Technical report, http://specs.xmlsoap.org/ws/2004/09/policy/ws-policyattachment.pdf, March 2006.

[BBDVf08] Georg Boenn, Martin Brain, Marina De Vos, and John ffitch. Automatic composition of melodic and harmonic music by answer set programming. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin Heidelberg, 2008.

[BBDvF11] Georg Boenn, Martin Brain, Marina De vos, and John Ffitch. Automatic music composition using answer set programming. *Theory Practice of Logic Programming*, 11(2-3):397–427, 2011.

[BDH07] Glenn Bruns, Daniel S Dantas, and Michael Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, FMSE '07, pages 12–21, New York, NY, USA, 2007. ACM.

[Bel77] N.D. Belnap. A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*. D. Reidel, Dordrecht, 1977.

[BF07] Jeremy W. Bryans and John S. Fitzgerald. Formal engineering of XACML access control policies in VDM++. In *ICFEM'07: Proceedings of the formal engineering methods 9th international conference on Formal methods and software engineering*, pages 37–56, Berlin, Heidelberg, 2007. Springer-Verlag.

[BH08] G. Bruns and M. Huth. Access-control via Belnap logic: Effective and efficient composition and analysis. In *21st IEEE Computer Security Foundations Symposium*, 2008.

[BHK+03] Don Box, Maryann Hondo, Chris Kaler, Hiroshi Maruyama, Anthony Nadalin, Nataraj Nagaratnam, Paul Patrick, Claus von Riegen, and John Shewchuk. Web services policy assertions language (WS PolicyAssertions). Technical report, http://xml.coverpages.org/ws-policyassertionsV11.pdf, March 2003.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical report, The Mitre Corporation, Bedford, MA, April 1977.

[BLD73] D.E. Bell, L.J. LaPadula, and United States. Air Force. Systems Command. Electronic Systems Division. *Secure Computer Systems: Mathematical Foundations.* National Technical Information Service, 1973.

[BMR04] Adam Barth, John C. Mitchell, and Justin Rosenstein. Conflict and combination in privacy policy languages. In *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 45–46, New York, NY, USA, 2004. ACM.

[BN89] D.F.C. Brewer and M.J. Nash. The chinese wall security policy. In *Proceedings on IEEE Symposium on Security and Privacy*, pages 206 –214, may 1989.

[BPCF73] D.E. Bell, L.J.L. Padula, Mitre Corporation, and United States. Air Force. *Secure Computer Systems: A mathematical model.* Secure Computer Systems. Mitre Corporation, 1973.

[BPSM+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. eXtensible Markup Language (XML) 1.0 (fifth edition). Technical report, http://www.w3.org/TR/xml/, November 2008.

[Bry05] Jeremy Bryans. Reasoning about XACML policies using csp. In *In SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35. ACM Press, 2005.

[BS03] S. Barker and P. J. Stuckey. Flexible access control policy specification with constraint logic programming. *TISSEC*, 6, 2003.

[CS95] P. Ciancarini and Leon Sterling. Report on the workshop: Applications of logic programming in software engineering. *The Knowledge Engineering Review*, 10:97–100, 3 1995.

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Communication ACM*, 19(5):236–243, May 1976.

[DeT02] John DeTreville. Binder, a logic-based security language. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 105 – 113, 2002.

[DFP09] Agostino Dovier, Andrea Formisano, and Enrico Pontelli. An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. *Journal of Experimental & Theoretical Artificial Intelligence*, 21(2):79–121, 2009.

[DLGHB+05] Giovanni Della-Libera, Martin Gudgin, Phillip Hallam-Baker, Maryann Hondo, Hans Granqvist, Chris Kaler, Hiroshi Maruyama, Michael Mcintosh, Anthony Nadalin, Nataraj Nagaratnam, Rob Philpott, Hemma Prafullchandra, John Shewchuk, Doug Walter, and Riaz Zolfonoon. Web services security policy language (WS SecurityPolicy). Technical report, http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf, July 2005.

[dVFJS07] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati. Access control policies and languages in open environments. In Ting Yu and Sushil Jajodia, editors, *Secure Data Management in Decentralized Systems*, volume 33 of *Advances in Information Security*, pages 21–58. Springer US, 2007.

[DWD11] Soma Shekara Sreenadh Reddy Depuru, Lingfeng Wang, and Vijay Devabhaktuni. Smart meters for power grid: Challenges, issues, advantages and status. *Renewable and Sustainable Energy Reviews*, 15(6):2736 – 2742, 2011.

[ET08] Esra Erdem and Ferhan Türe. Efficient haplotype inference with answer set programming. In Fox D. and Gomes C., editors, *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, pages 436–441. AAAI Press, 2008.

[FK92] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *15th National Computer Security Conference*, volume 1,

pages 554–563. National Institute of Standards and Technology, 1992.

[FKMT05] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.

[FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[GD72] G. Scott Graham and Peter J. Denning. Protection: Principles and practice. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, AFIPS '72 (Spring), pages 417–429, New York, NY, USA, 1972. ACM.

[GIL+10] Giovanni Grasso, Salvatore Iiritano, Nicola Leone, Vincenzino Lio, Francesco Ricca, and Francesco Scalise. An asp-based system for team-building in the gioia-tauro seaport. In Manuel Carro and Ricardo PeÃ±a, editors, *Practical Aspects of Declarative Languages*, volume 5937 of *Lecture Notes in Computer Science*, pages 40–42. Springer Berlin Heidelberg, 2010.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.

[GM03] Simon Godik and Tim Moses. eXtensible Access Control Markup Language (XACML) version 1.0 (OASIS Standard). Technical report, OASIS, https://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf, February 2003.

[Gol11] Dieter Gollmann. *Computer Security (3rd edition)*. John Wiley and Sons, Ltd, 2011.

[Gon89a] Li Gong. On security in capability-based systems. *SIGOPS Operting System Review*, 23(2):56–60, 1989.

[Gon89b] Li Gong. A secure identity-based capability system. In *In Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 56–63, 1989.

[GST+08] Martin Gebser, Torsten Schaub, Sven Thiele, Björn Usadel, and Philippe Veber. Detecting inconsistencies in large biological networks with answer set programming. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 130–144. Springer Berlin Heidelberg, 2008.

[HFK06] Vincent C. Hu, David F. Ferraiolo, and D. Rick Kuhn. Assessment of access control systems. Technical report, National Institue of Standards and Technology (NIST) Technology Administration U.S. Department of Commerce, Computer Security Division. Information Technology Laboratory. National Institute of Standards and Technology. Geitherburg, MD 20899-8930, September 2006.

[HNN09] Chris Hankin, Flemming Nielson, and Hanne Riis Nielson. Advice from belnap policies. *Computer Security Foundations Symposium, IEEE*, 0:234–247, 2009.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[HW03] Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. *Computer Security Foundations Workshop, IEEE*, 0:187, 2003.

[HW08] Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. *ACM Transaction on Information and System Security (TISSEC)*, 11(4):1 – 41, 2008.

[IMB+09] Harold Ishebabi, Philipp Mahr, Christophe Bobda, Martin Gebser, and Torsten Schaub. Answer set versus integer linear programming for automatic synthesis of multiprocessor systems from real-time parallel programs. *International Journal of Reconfigurable Computing*, 2009:6:1–6:11, 2009.

[Jac02] Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[Jim01] Trevor Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 106–115, Washington, DC, USA, 2001. IEEE Computer Society.

[JSSB97] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *In Proceedings of ACM SIGMOD International Conference on Management of Data*, 1997.

[KH07] Vladimir Kolovski and James Hendler. Xacml policy analysis using description logics. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, 2007.

[KHP07] Vladimir Kolovski, James Hendler, and Bijan Parsia. Formalizing xacml using defeasible description logics. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, 2007.

[Kol08] Vladimir Kolovski. Formal semantics of xacml, March 2008. Available at http://lists.oasis-open.org/archives/xacml/200310/pdf00000.pdf.

[Lam74] Butler W. Lampson. Protection. *SIGOPS Operating System Review*, 8(1):18–24, 1974.

[Lan81] Carl E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13:247–278, 1981.

[Lev84] Henry M. Levy. *Capability-Based Computer Systems.* Butterworth-Heinemann, Newton, MA, USA, 1984.

[LGF03] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security*, 6(1):128 – 171, 2003.

[LLL+12] Xu Li, Xiaohui Liang, Rongxing Lu, Xuemin Shen, Xiaodong Lin, and Haojin Zhu. Securing smart grid: cyber attacks, countermeasures, and challenges. *Communications Magazine, IEEE*, 50(8):38–45, August 2012.

[Llo84] John Wylie Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[LM03] Ninghui Li and John C. Mitchell. DATALOG with constraints: A foundation for trust management languages. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, PADL '03, pages 58–73, London, UK, UK, 2003. Springer-Verlag.

[LM13] Joohyung Lee and Yunsong Meng. Answer set programming modulo theories and reasoning about continuous changes. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[LMW02] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 114–130, 2002.

[LR06] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas? *ACM Transaction on Computational Logic*, 7(2):261–268, 2006.

[LTT99] Vladimir Lifschitz, LappoonR. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

[MGZ08] VeenaS. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.

[Mos05] Tim Moses. eXtensible Access Control Markup Language (XACML) version 2.0 (OASIS Standard). Technical report, OASIS, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, February 2005.

[MPT12] Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi. Formalisation and implementation of the XACML access control mechanism. In *Proceedings of the 4th International Conference on Engineering Secure Software and Systems*, ESSoS'12, pages 60–74, Berlin, Heidelberg, 2012. Springer-Verlag.

[NBG+01] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An a-prolog decision support system for the space shuttle. In *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages*, PADL '01, pages 169–183, London, UK, UK, 2001. Springer-Verlag.

[NBL09] Qun Ni, Elisa Bertino, and Jorge Lobo. D-algebra for composing access control policy decisions. In *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 298–309, New York, NY, USA, 2009. ACM.

[NGE11] J. Naruchitparames, M.H. Güne, and C.Y. Evrenosoglu. Secure communications in the smart grid. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 1171–1175, Jan 2011.

[Nie99] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[NIS12] NIST framework and roadmap for smart grid interoperability standards, release 2.0. Technical report, National Institue of Standards and Technology (NIST) Technology Administration U.S. Department of Commerce, Computer Security Division. Information Technology Laboratory. National Institute of Standards and Technology. Geitherburg, MD 20899-8930, February 2012.

[NS00] Ilkka Niemelä and Patrik Simons. Extending the smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, volume 597 of *The Springer International Series in Engineering and Computer Science*, pages 491–521. Springer US, 2000.

[NSP94] Nist special publication 800–7. Technical report, National Institute of Standards and Technology, October 1994.

[NSS99] Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics of weight constraint rules. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 1730 of *Lecture Notes in Computer Science*, pages 317–331. Springer Berlin Heidelberg, 1999.

[PKS10] P.P. Parikh, M.G. Kanabar, and T.S. Sidhu. Opportunities and challenges of wireless communication technologies for smart grid applications. In *Power and Energy Society General Meeting, 2010 IEEE*, pages 1–7, July 2010.

[PP89] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. Prentice Hall, 1st edition, 1989.

[Ris13] Erik Rissanen. eXtensible Access Control Markup Language (XACML) version 3.0 (OASIS Standard). Technical report, OASIS, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf, January 2013.

[RNN12] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. The logic of XACML. In Farhad Arbab and PeterCsaba Ölveczky, editors, *8th International Symposium on Formal Aspects of Component Software (FACS'11) Revised Selected Papers*, volume 7253 of *Lecture Notes in Computer Science*, pages 205–222. Springer Berlin Heidelberg, 2012.

[RNN13a] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. XACML 3.0 in answer set programming. In Elvira Albert, editor, *22nd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2012) Revised Selected Papers*, volume 7844 of *Lecture Notes in Computer Science*, pages 89–105. Springer Berlin Heidelberg, 2013.

[RNN13b] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. XACML 3.0 in answer set programming – extended version. Technical report, arXiv.org, February 2013.

[RNN14] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. The logic of XACML. *Science of Computer Programming*, 83(0):80–105, 2014. Formal Aspects of Component Software (FACS 2011 selected extended papers).

[SD01] P. Samarati and Sabrina De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *Foundations of Security Analysis and Design, Tutorial Lectures*, 2001.

[SN98] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, PADL '99, pages 305–319, London, UK, UK, 1998. Springer-Verlag.

[Ste91] Daniel F. Sterne. On the buzzword "security policy". In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 219–230, May 1991.

[TK06] Michael Carl Tschantz and Shriram Krishnamurthi. Towards reasonability properties for access-control policy languages. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, SACMAT '06, pages 160–169, New York, NY, USA, 2006. ACM.

[U.S85] U.S. Department of Defense. *DoD Trusted Computer System Evaluation Criteria*. DOD 5200.28.STD, 1985.

[VEK76] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, 1976.

[VNDCV06] Davy Van Nieuwenborgh, Martine De Cock, and Dirk Vermeir. Fuzzy answer set programming. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Logics in Artificial Intelligence*, volume 4160 of *Lecture Notes in Computer Science*, pages 359–372. Springer Berlin Heidelberg, 2006.

[VYR12] Roberto Vigo, Ender Yuksel, and Carroline Dewi Puspa Kencana Ramli. Smart grid security a smart meter-centric perspective. In *20th Telecommunications Forum (TELFOR)*, pages 127–130. IEEE, 2012.

[WL93] Thomas Y. C. Woo and Simon S. Lam. Authorization in distributed systems: A new approach. *Journal of Computer Security*, 1993.

[YZN+12] E. Yüksel, Huibiao Zhu, H.R. Nielson, Heqing Huang, and F. Nielson. Modelling and analysis of smart grid: A stochastic model checking case study. In *Sixth International Symposium on Theoretical Aspects of Software Engineering (TASE*, pages 25–32, July 2012.